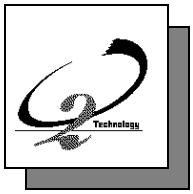



# $O_2$Look
# User Manual

**Release 5.0 - May 1998**

Information in this document is subject to change without notice and should not be construed as a commitment by $O_2$ Technology.

The software described in this document is delivered under a license or nondisclosure agreement.

The software can only be used or copied in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's own use.

Copyright 1992-1998 $O_2$ Technology.

All rights reserved. No part of this publication can be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopy without prior written permission of $O_2$ Technology.

$O_2$, $O_2$Engine API, $O_2$C, $O_2$DBAccess, $O_2$Engine, $O_2$Graph, $O_2$Kit, $O_2$Look, $O_2$Store, $O_2$Tools, and $O_2$Web are registered trademarks of $O_2$ Technology.

SQL and AIX are registered trademarks of International Business Machines Corporation.

Sun, SunOS, and SOLARIS are registered trademarks of Sun Microsystems, Inc.

X Window System is a registered trademark of the Massachusetts Institute of Technology.

Unix is a registered trademark of Unix System Laboratories, Inc.

HPUX is a registered trademark of Hewlett-Packard Company.

BOSX is a registered trademark of Bull S.A.

IRIX is a registered trademark of Siemens Nixdorf, A.G.

NeXTStep is a registered trademark of the NeXT Computer, Inc.

Purify, Quantify are registered trademarks of Pure Software Inc.

Windows is a registered trademark of Microsoft Corporation.

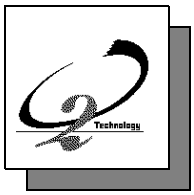
All other company or product names quoted are trademarks or registered trademarks of their respective trademark holders.

## Who should read this manual

This manual details how to use the $O_2$Look programming routines (functions and methods) in your methods and programs. It then describes how to customize your application interface and the $O_2$WidgetLook interface. This manual must be read together with either the $O_2$C Reference manual, ODMG C++ Binding Guide, or the C interface to $O_2$ Manual.

Other documents available are outlined, click below.

**See O2 Documentation set.**




# TABLE OF CONTENTS

This manual is divided into the following chapters:


- 1 - Introduction
- 2 - Methods and functions
- 3 - Masks
- 4 - Resources
- 5 - Widget Look

# TABLE OF CONTENTS


O2Look i

Who should read this manual ... iii
See O2 Documentation set. ... iii

1 Introduction 11

1.1 O2 Product Line ... 12

1.2 Features and advantages ... 14

Easy to update and user-friendly ... 14
Easy to use and customize ... 14
Open and Integrated ... 15

1.3 Presentations ... 16

1.4 Drag and Drop ... 18

1.5 Programming with O2Look ... 19

Methods ... 19
Functions ... 20
Masks and Resources ... 20

1.6 User Manual overview ... 21

2 Methods and Functions 23

2.1 Introduction ... 24

System-defined C types ... 24
O2C options ... 25

2.2 Display methods ... 25

all_presentations ... 27
consult ... 28
disable_method ... 29
display ... 31
edit ... 33
enable_method ... 35
erase_all ... 36
present ... 37
refresh_all ... 38

rubout ..... 39

**2.3 Callback methods ..... 40**

bitmap ..... 41
erase ..... 43
menu ..... 44
save ..... 46
title ..... 47

**2.4 Library functions ..... 49**

lk_all_mapped_presentations ..... 50
lk_all_presentations ..... 51
lk_consult ..... 52
lk_current_presentation ..... 54
lk_delete_all ..... 56
lk_delete_presentation ..... 57
lk_epilogue ..... 58
lk_free ..... 59
lk_grab ..... 60
lk_map ..... 61
lk_present ..... 64
lk_prologue ..... 66
lk_refresh_presentation ..... 68
lk_unmap ..... 70
lk_wait ..... 71

**3 Masks 73**

**3.1 Introduction ..... 74**

Mask Functions ..... 74
Resource specification in Functions ..... 76

**3.2 Functions ..... 79**

lk_atom ..... 80
lk_destroy_mask ..... 82
lk_generic ..... 83
lk_list ..... 85
lk_method ..... 87
lk_object ..... 89

lk_protected....92
lk_set....94
lk_specific....96
lk_tuple....97

4 Graphic Resources 101

4.1 Using graphic resources....102
4.2 Resource files....102
Special resources for application icons....105
4.3 Resource arrays....106
4.4 How to read resource tables....107
Name....108
Type....108
Default....110
Access....110
4.5 Global resources....111
4.6 Presentation resources....113
4.7 Graphic resources for integers....115
4.8 Graphic resources for real numbers....116
4.9 Graphic resources for characters....117
4.10 Graphic resources for strings....118
4.11 Graphic resources for boolean values....119
4.12 Graphic resources for lists....121
4.13 Graphic Resources for sets....125
4.14 Graphic resources for tuples....129
4.15 Graphic Resources for Objects....131

# TABLE OF CONTENTS


5 Widget Look 137

5.1 Introduction ..... 138

5.2 Tools ..... 138

Xo2Look ..... 139
Xo2CreateLook ..... 143
Xo2SetO2Value ..... 144
Xo2ConsultO2Value ..... 146

INDEX 147

# TABLE OF CONTENTS

# 1 Introduction

YOUR GUI GENERATOR

Congratulations! You are now a user of $O_2$Look - a tool for the design and development of the Graphical User Interface of your application.

To face the technical challenge of designing and implementing a Graphical User Interface, $O_2$Look provides software developers with a powerful integrated tool enabling the on-screen graphical display and interactive manipulation of complex and multimedia objects and involving no programming and no drawing.

This introductory chapter is divided as follows:

- O2 Product Line
- Features and advantages
- Presentations
- Drag and Drop
- Programming with O2Look
- User Manual overview

# 1 Introduction

## 1.1 $O_2$ Product Line

The system architecture of $O_2$ is illustrated in Figure 1.1..

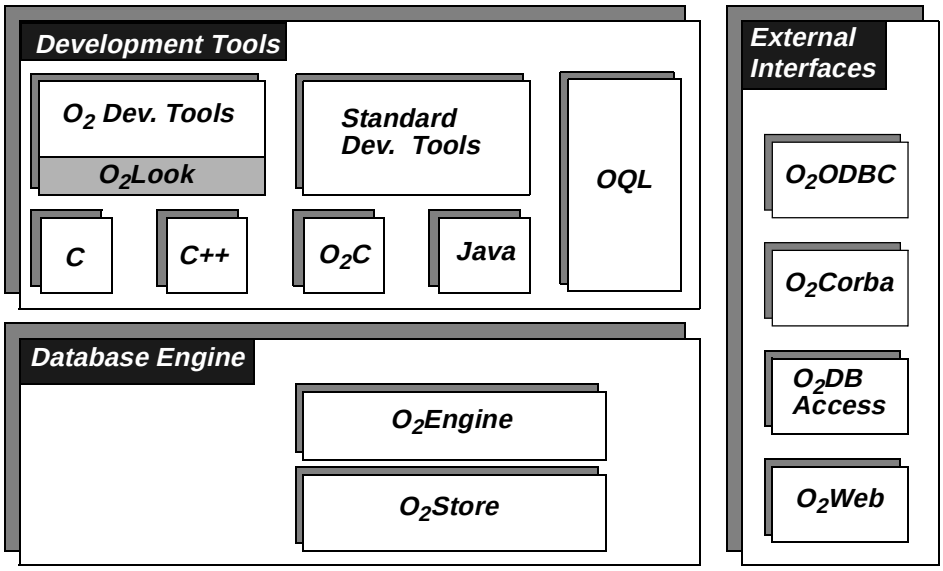



*Figure 1.1: $O_2$ System Architecture*

The $O_2$ system can be viewed as consisting of three components. The *Database Engine* provides all the features of a Database system and an object-oriented system. This engine is accessed with *Development Tools*, such as various programming languages, $O_2$ development tools and any standard development tool. Numerous *External Interfaces* are provided. All encompassing, $O_2$ is a versatile, portable, distributed, high-performance dynamic object-oriented database system.

Database Engine:

- $O_2$Store — The database management system provides low level facilities, through $O_2$Store API, to access and manage a database: disk volumes, files, records, indices and transactions.
- $O_2$Engine — The object database engine provides direct control of schemas, classes, objects and transactions, through $O_2$Engine API. It provides full text indexing and search capabilities with $O_2$Search and spatial indexing and retrieval capabilities with $O_2$Spatial. It includes a Notification manager for informing other clients connected to the same $O_2$ server that an event has occurred, a Version manager for handling multiple object versions and a Replication API for synchronizing multiple copies of an O2 system.

Programming Languages:

$O_2$ objects may be created and managed using the following programming languages, utilizing all the features available with $O_2$ (persistence, collection management, transaction management, OQL queries, etc.)

- C — $O_2$ functions can be invoked by C programs.
- C++ — ODMG compliant C++ binding.
- Java — ODMG compliant Java binding.
- $O_2$C — A powerful and elegant object-oriented fourth generation language specialized for easy development of object database applications.
- OQL — ODMG standard, easy-to-use SQL-like object query language with special features for dealing with complex $O_2$ objects and methods.

$O_2$ Development Tools:

- $O_2$Graph — Create, modify and edit any type of object graph.
- $O_2$Look — Design and develop graphical user interfaces, provides interactive manipulation of complex and multimedia objects.
- $O_2$Kit — Library of predefined classes and methods for faster development of user applications.
- $O_2$Tools — Complete graphical programming environment to design and develop $O_2$ database applications.

Standard Development Tools:

All standard programming languages can be used with standard environments (e.g. Visual C++, Sun Sparcworks).

External Interfaces:

- $O_2$Corba — Create an $O_2$/ Orbix server to access an $O_2$ database with CORBA.
- $O_2$DBAccess — Connect $O_2$ applications to relational databases on remote hosts and invoke SQL statements.
- $O_2$ODBC — Connect remote ODBC client applications to $O_2$ databases.
- $O_2$Web — Create an $O_2$ World Wide Web server to access an $O_2$ database through the internet network.

## 1.2 Features and advantages

$O_2$Look enables you to design and create the graphical user interface of your object-oriented database applications.

It is unique because it automatically creates the graphical display of any database object.

It offers software developers a powerful integrated tool enabling the on-screen graphical display and interactive manipulation of complex and multimedia objects, and involves no drawing and no programming.

Some of its features include:

- direct drag and drop operations on objects.
- the application can control the interactive dialog between the application and the end user.
- $O_2$Look automatically creates a menu from which object methods can be triggered.
- you can easily customize the $O_2$Look presentation to meet your specific needs.
- links can be created between presentation windows for dispatching actions (like moving, iconifying, saving).

### Easy to update and user-friendly

You can test your $O_2$Look graphical user interface immediately giving you extremely fast prototyping facilities.

Any changes you make to your application can be reflected in the interface at a very low cost and very easily.

This enables your end application to be extremely user-friendly because its graphical user interface is coherent and user-friendly and can be fully customized to meet the specific needs and expectations of all your end-users.

### Easy to use and customize

You need no programming experience to use the toolkit since development with $O_2$Look is code free.

The graphical presentation of any object in the database is automatically built, irrespective of its complexity, by reading its class definition. $O_2$Look automatically creates a menu from which object methods can be triggered and which pops up a dialog box for entering parameters.

About twenty predefined methods permitting the display and the manipulation of presentations are attached to the class Object and inherited by any subclass.

Although $O_2$Look automatically generates generic presentations of objects and values that can be used immediately, you can easily customize these presentations to meet your own requirements. You can also use tailor-made presentations.

For the easy customization of an object's presentation, you have a Mask mechanism that enables you to define which parts of an object are visible. Masks also give you the use of the text, bitmap, image, and widget editors[1] used in conjunction with $O_2$Kit.

A Resources mechanism describes the graphical aspect (e.g. color, font), the menu and the editability of an object. You can modify resources without re-compiling your application.

Your end-users can modify resources not specified directly in the application by the developer.

## Open and Integrated

Your $O_2$Look presentations are displayed with OSF/ Motif look and feel.

For more specific needs you can use $O_2$Look in conjunction with other graphic tools such as $O_2$Xt and $O_2$WidgetLook.

$O_2$Look is fully integrated with $O_2$Engine in C, C++ and $O_2$Engine API.

When used with $O_2$C, $O_2$Look constitutes a powerful 4th generation object-oriented language.

1. Also the graph editor of $O_2$Graph.

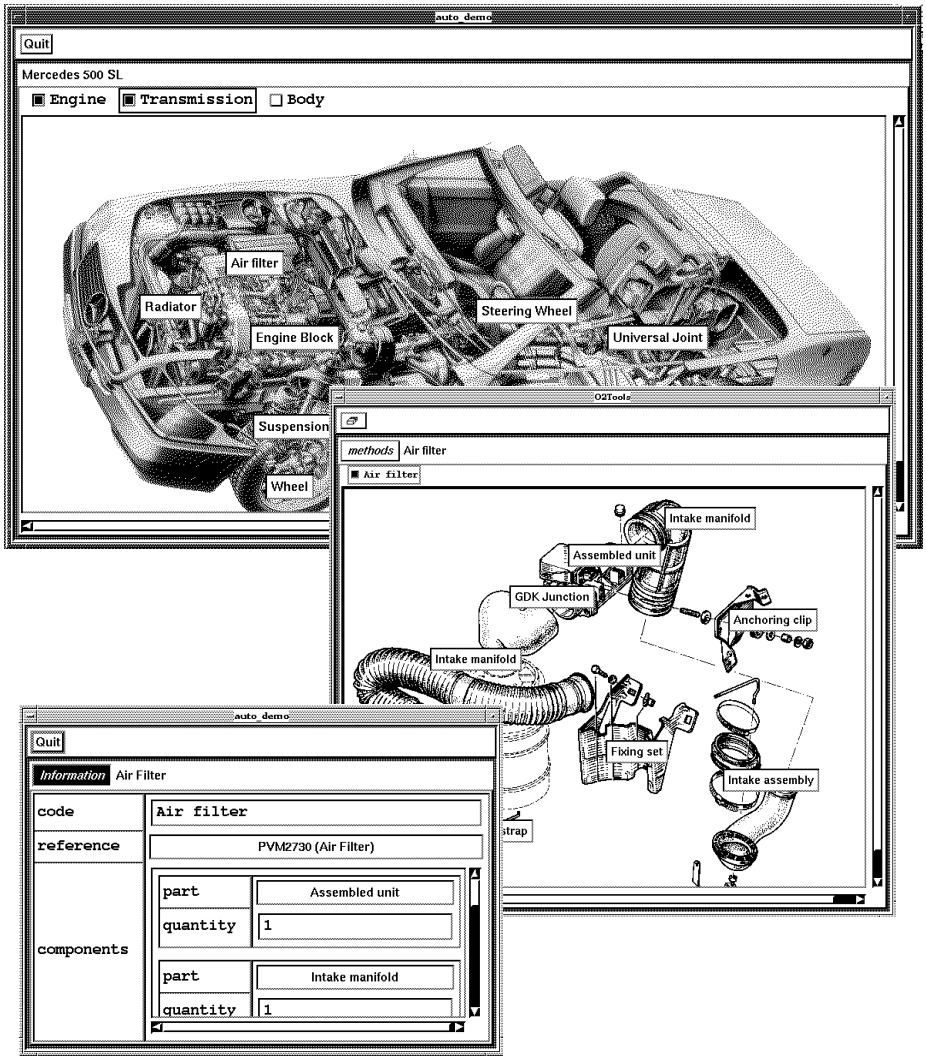



*Figure 1.2: $O_2$Look presentations*

## 1.3 Presentations

$O_2$Look is a graphical user interface for the $O_2$ object-oriented database management system. It operates under the X Window System, uses of the Motif widget library, and conforms to the recommendations of the Motif style guide.

Functionally, $O_2$Look is a Motif extension dedicated to the screen display and manipulation of $O_2$ objects and values.

You can display any $O_2$ object or value, regardless of its data type, in a re-sizable Motif window called a presentation.

For example, consider a class **Person** defined as follows:

```
class Person
 public type tuple (name: string,
                    age: integer,
                    position: string,
                    spouse: Person,
                    children: list(Person))

 method public birthday: integer

 end;
```

With $O_2$Look, you use various methods and functions to display an object of this class in the following type of window shown in Figure 1.3.

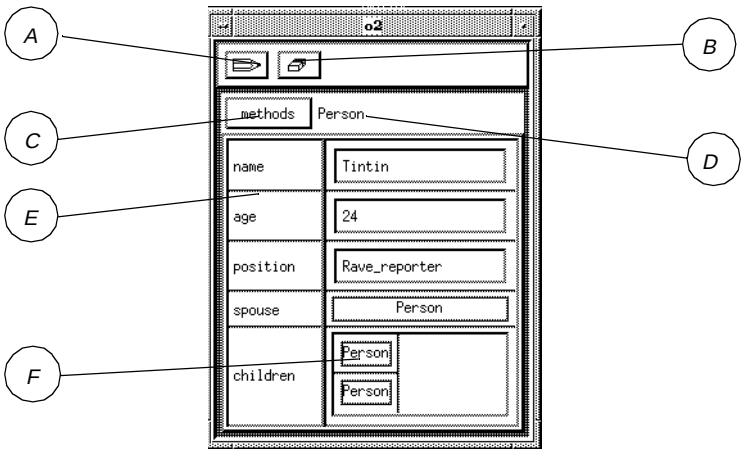



*Figure 1.3: $O_2$Look Presentation*

*A: Pencil button* *B: Eraser button*
*C: Public methods menu* *D: Presentation title*
*E: Object contents* *F: Component object icon*

If you are allowed to carry out modifications in the window there is always a pencil (*A*) and an eraser (*B*) button.

If you can only read the contents of the window, there is only an eraser button.

To remove the display, click on the Pencil or the Eraser button. To save your modifications, click on Pencil. The value is updated and the presentation disappears. If you click on the Eraser all modifications are ignored and the presentation disappears[1].

The public methods menu appears when you click the left mouse button on the menu button (*C*) of an object presentation.

The presentation title (*D*) is, by default, the class name of the object being displayed (and is therefore blank for values). You can change the presentation title from the default class title of the object. If the presentation shows an object that is a component of another object (*F*), each object has its own pop-up menu which is displayed by clicking on the Object icon using the right mouse button.

**Note**

The presentation of a value, of course, has no public methods menu (*C*) and no title (*D*).

## 1.4 Drag and Drop

The $O_2$Look drag and drop mechanism allows you to copy or move a selected element to a different location.

When you copy an element, the original remains unchanged and the copy is pasted into the new location.

When copying elements, if the type is a value, you can copy the value, but if the type is an object, you can only share the object.

When you move an element, the original element is cut from the source and is pasted into the new location.

1. You control the appearance and utilization of these two buttons using the lk_wait and lk_grab functions described in Section 2.3.

To drag an element, select it using the middle mouse button and drag it to the location of your choice.

Be careful when selecting and dragging elements that you are actually allowed to copy and move these elements, and that the destination will accept the elements in question:

- When you drag an element that cannot be cut from the source, the operation is automatically a copy operation, e.g, dragging a tuple attribute or an item of a non-editable collection.
- If it can be cut from the source, the operation is automatically a move operation, e.g, dragging an item from an editable collection.
- Pressing the "Ctrl" key when dragging implies a copy operation even if the element can be cut.

## 1.5 Programming with $O_2$Look

You create and manipulate presentations using various tools supplied with $O_2$Look.

### Methods

The most basic $O_2$Look operation on an $O_2$ object is to invoke a system-defined method on it.

For example, to create a presentation that displays the value of the object **`tintin`** and then waits for the end-user to acknowledge it, you simply use the following $O_2$C code:

```
tintin->display;
```

All the various system-defined methods you can use to generically display objects are given in Section 2.3. Also given are the $O_2$Look callback methods that control some part of the aspect and behavior of presentations. This includes the bitmap image used for an icon, or what happens when the Pencil button is clicked.

# 1 Introduction

## Functions

To build and manipulate presentations for values, you invoke an $O_2$Look function (as values have no methods). This function library is given in Section 2.4.

Several of these functions simply repeat the operations of the methods (sometimes with more sophisticated options). If you use these functions with their default parameters, it is often the simplest way to construct a graphic application quickly and cleanly.

You can use the $O_2$Look library functions to display and manipulate objects as well as values. In fact for highly specialized graphic applications, you need the elaborate options of these functions as they supply a flexible means to define and manipulate presentations.

One very important way is to specify presentation masks.

## Masks and Resources

A mask definition describes a template for the format of a presentation: which tuple attributes are to be displayed, for example, and which can be modified. Most $O_2$Look operations use default masks.

However, you can find that the standard masks are not suitable for all situations. You have control over the graphic resources used by the system to build windows.

Graphic resources include color, fonts, icon and button bitmaps, and whether or not the displayed data can be modified.

$O_2$Look resources are an extension of X Window and Motif resources. Some $O_2$Look functions return error codes, but these are useful mainly to ascertain whether everything went well or not.

Run-time errors are handled by $O_2$Look itself; an appropriate error message is displayed on the screen. If the error is recoverable (for example, the specification of a mask inappropriate for the value being displayed), a “Warning” or “Attention” message is displayed, and a default action is taken (use of a default mask, for example).

If the error is serious, an “Error” message is displayed and the $O_2$Look function terminates immediately with a non-zero returned value.

## 1.6 User Manual overview

The $O_2$Look User manual is divided up into the following chapters:

- ***Chapter 1 - Introduction***

  This chapter introduces the features and advantages $O_2$Look offers you. It then introduces the basic concepts of $O_2$Look Presentations and overviews how to program your application using $O_2$Look using the system-supplied methods, functions, masks and graphic resources. Finally it gives the system-defined C types you need to use $O_2$Look.

- ***Chapter 2 - Methods and Functions***

  This chapter introduces the system defined C types and options $O_2$C you need when using $O_2$Look methods and functions to write your graphic applications. It then gives the syntax of all the $O_2$Look display and callback methods, and all the $O_2$Look library functions you need to display and manipulate any object or value.

- ***Chapter 3 - Masks***

  This chapter describes how to use all the various $O_2$Look masks.

- ***Chapter 4 - Graphic Resources***

  This chapter describes how to use all the various $O_2$Look graphic resources and gives all the relevant resource tables.

- ***Chapter 5 - Widget Look***

  This chapter describes the $O_2$WidgetLook. This is an X Toolkit compatible user-interface component and contains all the power of $O_2$Look in a simple widget.

# 1 Introduction

# 2 Methods and Functions

This chapter outlines the following $O_2$Look methods and functions:

- Introduction
- Display methods
- Callback methods
- Library functions

# 2 Methods and Functions

## 2.1 Introduction

This section introduces the system defined C types and $O_2$C options you need when using $O_2$Look methods and functions to write your graphic applications.

### System-defined C types

The system defines several C types:

- ***Lk_presentation***

  The **Lk_presentation** type is associated to $O_2$Look presentation windows. Each presentation definition is an instance of this type.

- ***Lk_mask***

  This type is associated to display masks used within presentations. Each display mask is an instance of this type.

- ***Lk_attribute_mask***

  You use this type to specify the display mask of an individual tuple attribute. It is used only with the **lk_tuple** mask function.

```
typedef struct {
   char *name;
   Lk_mask mask;
} Lk_attribute_mask;
```

- ***Lk_resource***

  This type is associated with $O_2$Look graphic resources. Each resource is specified with an instance of this type:

```
typedef struct {
   char *name;
   char *value;
} Lk_resource;
```

### $O_2$C options

The type definitions, described above, as well as a number of enumerated types used in the Library functions, are found in a header file called **o2_look.h** in the **include** subdirectory of the $O_2$ installation directory.

This header file is automatically included by the $O_2$C compiler for all $O_2$C methods, programs and functions. If this causes a problem, in that for example, you have an application that uses a definition that conflicts with the **o2_look.h** definition, simple use the **-nolook** option of the set options schema command. See the $O_2$C Reference manual for further details.

Note that the **o2_look.h** header file and its definitions are used only by $O_2$C code.

You may often want to design methods and programs that take presentations or masks as Arguments, or return them as returned values.

You should therefore treat presentations and masks as integers in $O_2$ data definition language commands, including method and program signatures, and cast as type **Lk_presentation** or **Lk_mask** within method code.

## 2.2 Display methods

The following methods are attached to class **Object** and can therefore be sent to any object.

They enable you to display and manipulate graphic representations of objects on the screen.

# 2 Methods and Functions

Only the most generic of $O_2$Look functions are performed by these system-supplied methods.

For a more specific use, or for the display and manipulation of values as well as objects, use the Library functions.

None of the methods described in this chapter function in alphanumeric mode as they all require $O_2$Look.

## *Note*

The **display** method described below functions in both modes, although it has a particular behavior in $O_2$Look.

The following system-supplied display methods are described:

- **all_presentations**
- **consult**
- **disable_method**
- **display**
- **edit**
- **enable_method**
- **erase_all**
- **present**
- **refresh_all**
- **rubout**

## all_presentations

**Summary** Returns a list of all receiver object presentations.

**Syntax** *receiver*`->all_presentations`

**Arguments** None.

**Returns** A value of type **`list (integer)`**. Each element in the list corresponds to a presentation. If there are no presentations, the empty list is returned.

**Description** The **`all_presentations`** method lists all the receiver object presentations defined for the current application.

This includes those presentations in which the receiver object appears as a component as well as those presentations which are devoted to the receiver object, and irrespective of whether the presentations are mapped or unmapped.

***Note***

If the receiver object appears inside a specific editor, that presentation is not included in the list.

**Example**

```
o2 list (integer) lp;
...
lp = (o2 list(integer)) client->all_presentations;
```

## consult

**Summary** Assigns a new value to the receiver object.

**Syntax** *receiver*`->consult (`*presentation*`)`

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

**Description** The **consult** method takes the current value from the $O_2$Look presentation identifier in the *presentation* argument, dynamically checks the types, and then assigns this value to the receiver object.

**Returns** Zero if all went well, or non-zero integer if the *presentation* given does not exist.

**Example**

```
selected_hotel->consult (p);
```

## disable_method

**Summary** Disables a specified method.

**Syntax** *receiver*`->disable_method (`*method_name*`)`

**Arguments**

*method_name* The name of the method to be disabled. This is character string which can be a variable defined as **`char *`**, an $O_2$ **`string`**, or a string literal between double-quotes.

**Description** The **`disable_method`** method disables the method *method_name* in the menu of public methods attached to receiver object presentations, a menu which lists all public methods defined in or inherited by the object class.

The method is immediately disabled for all the presentations of the receiver object currently defined. The method name is shadowed on the menu.

To reverse the **`disable_method`** method, refer to the **`enable_method`** method.

Any presentation defined after the **`disable_method`** is unaffected.

**Returns** Zero if the method is successful and a non-zero if the method name specified is not a public method of the receiver object class.

**Example**

```
o2 Person this_employee = new Person;
Lk_presentation p = this_employee->present(0);
this_employee->disable_method ("view_salary");
lk_map (p, LK_MANUAL, LK_FREE, LK_SCREEN, 0, 0);
lk_grab(p);
```

In the menu of public methods, **`view_salary`** is now shadowed and you cannot select it.

# 2 Methods and Functions

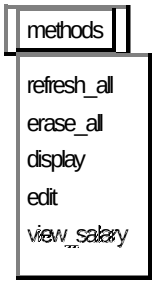



*Figure 2.1: Methods menu with shadowed item*

## display

**Summary** Displays the receiver object value.

**Syntax** *receiver* **-> display**

**Arguments** None

**Description** The **display** method displays the public and/ or read-only value of the receiver object. You cannot modify the presentation contents and consequently, there is no Pencil button.

**Returns** Nothing

**Example**

```
run body {
o2 Person p = new Person, q = new Person,
          r = new Person, s = new Person;
p -> name = "Tintin";
p -> age = 24;
p -> position = "Rave_reporter";
p -> spouse = q;
p -> children = list (r, s);
p -> display;
}
```

If you are using $O_2$Look, you see the window shown in Figure 2.2.:

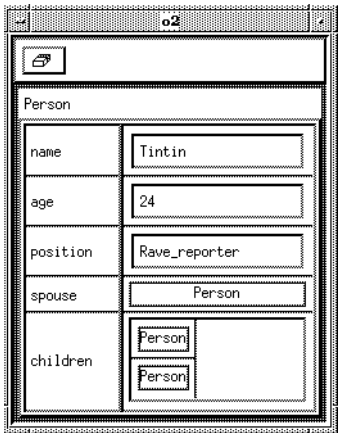



*Figure 2.2: Display method in $O_2$Look*

## edit

**Summary** Displays the receiver object value

**Syntax** *receiver*`->edit`

**Arguments** None

**Returns** **LK_SAVE** – if the Pencil button is clicked, or

**LK_ERASE** – if the Eraser button is clicked.

**Description** The **edit** method is the same as the **display** method described in that it displays the receiver object value on screen. The difference is that you can modify the contents of the presentation window and there are both Pencil and Eraser buttons. Only public properties are displayed.

**Example**

```
run body {
o2 Person p = new Person, q = new Person,
              r = new Person, s = new Person;
p -> name = "Tintin";
p -> age = 24;
p -> position = "Rave_reporter";
p -> spouse = q;
p -> children = list (r, s);
p -> edit;
}
```

You obtain the following presentation shown below in Figure 2.3.

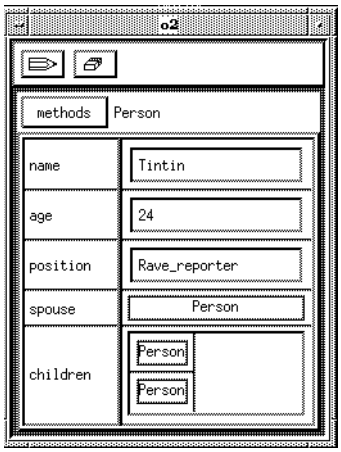



*Figure 2.3: Presentation after an edit method*

To remove the display, click on the Pencil or the Eraser button, shown in Figure 2.3.

If you click on Pencil the displayed value and any changes you have made are assigned to the receiver object. If you click on Eraser any changes you made are ignored.

## enable_method

**Summary** Reverses the **disable_method**.

**Syntax** *receiver*->**enable_method (***method_name***)**

**Arguments**

| | |
|---|---|
| *method_name* | The name of the method to be enabled. This is a character string which can be a variable defined as **char ***, an $O_2$ **string**, or a string literal enclosed in double-quotes. |

**Description** The **enable_method** method reverses **disable_method** and restores the previously disabled method *method_name* in the menu of the receiver object presentations.

**Returns** Zero if successful or a non-zero integer if the specified method name is not one of the public method belonging to the object class.

**Example**

```
Lk_presentation p = this_employee->present(0);
this_employee->disable_method ("view_salary");
...
this_employee->enable_method ("view_salary");
```

In the menu of public methods, **view_salary** is now enabled (it is no longer shadowed) and you can select it.

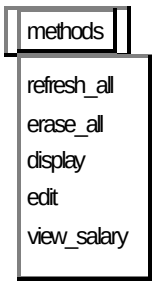



*Figure 2.4: Methods menu with shadowed item*

## erase_all

**Summary** Removes all receiver object presentations.

**Syntax** *receiver*`->erase_all`

**Arguments** None.

**Description** The **`erase_all`** method removes all the receiver object presentations defined in the current application.

Mapped presentations are erased from the screen, and the definitions of both mapped and unmapped presentations are deleted. If the object is a component of the presentations of another object or value, these presentations are also erased.

***Note***

In the present version of $O_2$Look, if the receiver object appears inside a specific editor, that presentation is not erased.

**Returns** Zero if successful or a non-zero integer if one or more of the presentations is locked (with an **`lk_grab`** or **`lk_wait`** function) and cannot be erased.

**Example**

```
client->erase_all;
```

## present

**Summary** Builds a presentation of the receiver object.

**Syntax** *receiver*->**present (***mask***)**

**Arguments**

| | |
|---|---|
| *mask* | The mask you want to use to build the presentation is a value of type **Lk_mask**. If left zero, the method takes the default mask **lk_protected**. |

**Description** The **present** method builds an object presentation according to its class declaration. This method does not display the object but returns a presentation identifier which you can then use in other $O_2$Look functions.

You can specify a display mask (a value of type **Lk_mask**) to use in the presentation. Otherwise, a default mask is used. In this way the **present** method is more flexible than the **display** method.

**Returns** The presentation identifier which is a value of type **Lk_presentation**.

**Example**

```
Lk_presentation p;
o2 Client current_client;

...

p = current_client->present (0);
lk_map (p, LK_COORDINATE, 0, 0, 200, 0);
lk_grab (p);
lk_delete_presentation (p);
```

## refresh_all

**Summary** Refreshes all screen representations of the receiver object using the current value of that object.

**Syntax** *receiver*`->refresh_all`

**Arguments** None

**Description** The **`refresh_all`** method refreshes all screen representations of the receiver object using the current value of that object.

This method affects all the presentations defined using the receiver object, whether as a whole or as a component part of another object or value, and whether or not the presentation is actually mapped anywhere.

If the receiver object appears as a component of a larger presentation, only the component related to the receiver object is refreshed.

### *Note*

In the present version of $O_2$Look, if the receiver object appears inside a specific editor, that presentation is *not* refreshed.

**Returns** Nothing

**Example**

```
client->refresh_all;
```

## rubout

**Summary** Replaces all receiver object presentations with nil object representations.

**Syntax** *receiver*`->rubout`

**Arguments** None.

**Description** The **rubout** method replaces all representations of the receiver object with nil object representations thereby removing all references to the object in all mapped and unmapped presentations.

The replacement takes place only at the presentation level. The object is not deleted from the database and no references to it are actually changed in the database.

However, if you issue a **consult** method on the presentations involved, or if your end-user clicks on the Pencil button, the changes are recorded in the database.

### Note

In the present version of $O_2$Look, if the receiver object appears inside a specific editor, this representation of it is *not* replaced with a nil object.

**Returns** Zero if successful or a non-zero integer if one or more of the presentations are locked (using the **lk_grab** or **lk_wait** function) and cannot be erased.

**Example**

```
client->rubout;
```

## 2.3 Callback methods

There are a number of generic $O_2$Look methods automatically invoked by $O_2$Look and are rarely explicitly invoked.

For example, a **save** method is triggered whenever the end-user clicks on the Pencil button in an unlocked object presentation.

You can redefine these internal $O_2$Look methods for any class at any level in the class hierarchy and thereby customize the way in which $O_2$Look proceeds in certain situations.

If any of these methods are redefined to refer to an object other than **self**, then that object must exist when the method is triggered. Otherwise a nil object exception is raised.

The following methods are described below:

- **bitmap**
- **erase**
- **menu**
- **save**
- **title**

## bitmap

**Summary** Defines a bitmap used to represent the receiver object as an icon.

**Syntax** *receiver*`->bitmap`

**Arguments** None

**Description** If you want to represent the receiver object as an icon, you use **bitmap** to define a bitmap.

It is automatically triggered when you use the **present**, **display** or **edit** methods, or the **lk_present** function.

The bitmap is empty by default and the icon is simply the class name of the object centered in a box.

You can redefine the method to return any bitmap.

The $O_2$Look resource **bitmap** overrides the **bitmap** method, and if it is activated the method is not triggered.

**Returns** A tuple of the following type:

```
tuple (width: integer,
       height: integer,
       bitsmap: bits)
```

The attribute **width** is the horizontal size of the icon (in pixels), **height** is the vertical size of the icon (in pixels), and **bitsmap** is the icon image in X Windows bitmap format.

**Example** If you have a class **Person**, you can represent each object, i.e. each person with a photo.

A scanned photograph in X bitmap format is used for the icon, and one of the attributes of class **Person** is **photo** itself an object of the system-supplied class **Bitmap**.

You must redefine the bitmap method of the class **Person** as follows:

```
return *(self->photo);
```

You must assign the object attribute **photo** before displaying the **Person** object that contains it. Otherwise, when $O_2$Look triggers the

**bitmap** method for the display and the method gets a nil object exception.

For example:

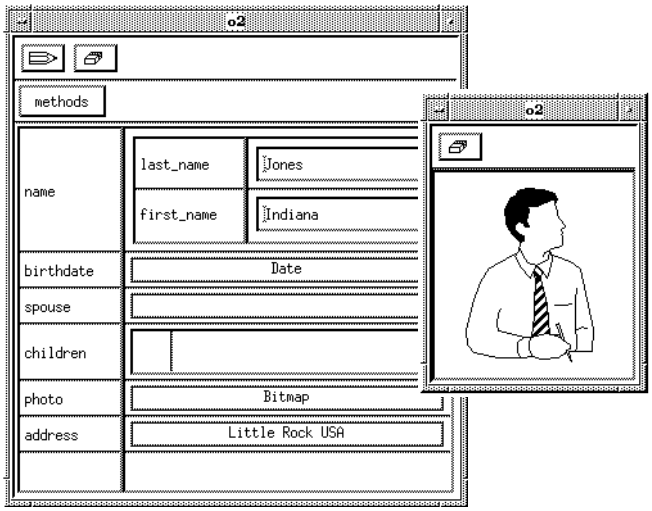



*Figure 2.5: Bitmap image*

## erase

**Summary** Defines the behavior of the receiver object presentation when the Eraser button on this presentation is clicked by the end-user.

**Syntax** *receiver*->**erase** (*presentation*)

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

**Description** The **erase** method defines the behavior an unlocked object presentation when the Eraser button is clicked.

This method is automatically triggered when the end-user clicks on the Eraser button:

The **erase** method invokes, by default the **lk_delete_presentation** function on the presentation.

You can redefine the **erase** method to carry out any other operation.

**Returns** Nothing

**Example** You can redefine the **erase** method in the following way:

```
{for (p in self->all_presentations)
   {lk_delete_presentation(p);
   }
}
```

## menu

**Summary** Defines the public methods menu.

**Syntax** *receiver*`->menu`

**Arguments** None

**Description** The **menu** method defines the menu of public methods that appears whenever you click on the right mouse button on a receiver object icon, or the left mouse button on the Menu button of an expanded object presentation.

This method is automatically triggered when you define such a presentation using the **present**, **display** or **edit** methods, or the **lk_present** function. By default all public methods are in the menu but you can redefine the menu method to return any selection of public methods available.

The $O_2$Look resource **menu** overrides the **menu** method, and if it is activated the method is not triggered.

If the presentation is refreshed, the **menu** method (and **menu** resource) is not taken into account thereby ensuring that a menu for a particular presentation cannot be modified without destroying and redefining that presentation.

However, you can disable and enable menu items at any time.

**Returns** A list of strings. By default a list of strings of the names of all user-defined public methods attached to the receiver object class, and the system-defined methods **display** and **edit**.

**Example** You can redefine the **menu** method for a class **Country** so that it only shows the system-defined method **display**.

This means that your application end-users cannot modify the contents of a **Country** object, because they cannot use any other public methods.

This is shown overleaf.

```
method body menu: list(string) in class Address {
 return list ("display");
}
```

when you click on the Address icon, in this case the Class name, using the right mouse button you get a menu containing the display item as shown in (*A*) Figure 2.6:

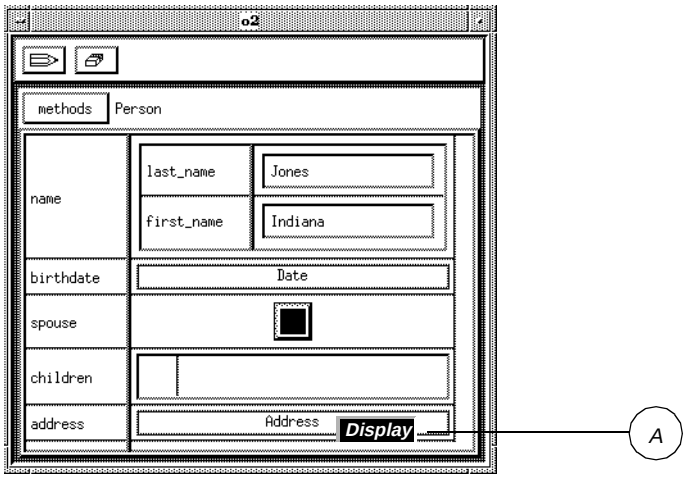



*Figure 2.6: A modified menu*

## save

**Summary** Defines the behavior of the receiver object presentation when the presentation Pencil button is clicked.

**Syntax** *receiver*`->save (`*presentation*`)`

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

**Description** The **save** method defines the behavior of an unlocked receiver object presentation when the end-user clicks on the Pencil button:

It is automatically triggered by $O_2$Look.

By default, the **save** method invokes the **lk_consult** function on the presentation concerned.

You can redefine the save method to carry out any other operation.

**Returns** Nothing

**Example** You can redefine the **save** method in the following way:

```
{lk_consult (p, self);
  self->refresh_all;
  lk_delete_presentation(p);
}
```

## title

**Summary** Defines a string that is in the title of any receiver object presentation

**Syntax** *receiver*->**title**

**Arguments** None

**Description** The **title** method defines a string used in the title of any receiver object presentation. This method is automatically triggered when you define a presentation using the **present**, **display** or **edit** method, or **lk_present** function. By default, the string used for the presentation title is the receiver object class name.You can redefine this method to return any other string. The $O_2$Look resource **titleString** overrides the **title** method and if it is active the method is not triggered.

**Returns** The string used in the presentation title. By default the name of the class.

**Example** You can redefine the **title** method in the class **Address** in order to show the name of the city and the country in the presentation of objects of the class **Address**. Otherwise, only the class name **Address** is displayed. Use the following method body:

```
method body title: string in class Address
{
  return self->city+ " " + self->country;
}
```

This gives the following type of window shown in (*A*) Figure 2.7:

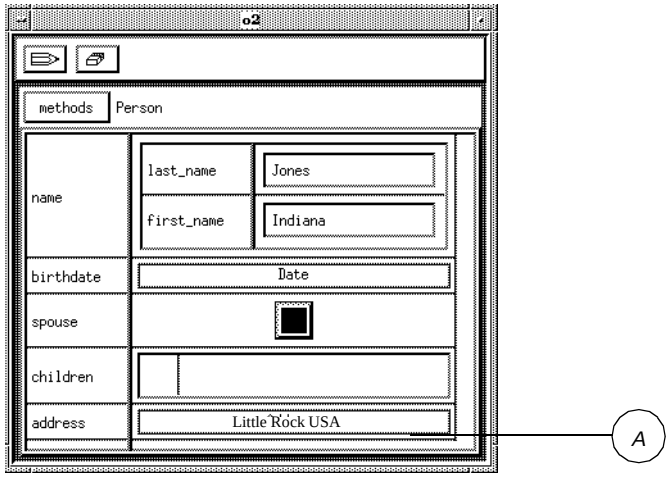



*Figure 2.7: A modified class title*

## 2.4 Library functions

This section describes the following $O_2$Look library functions:

- **lk_all_mapped_presentations**
- **lk_all_presentations**
- **lk_consult**
- **lk_current_presentation**
- **lk_delete_all**
- **lk_delete_presentation**
- **lk_epilogue**
- **lk_free**
- **lk_grab**
- **lk_map**
- **lk_present**
- **lk_prologue**
- **lk_refresh_presentation**
- **lk_unmap**
- **lk_wait**

## lk_all_mapped_presentations

**Summary** Lists all the mapped presentations in the current application.

**Syntax** **`lk_all_mapped_presentations()`**

**Arguments** None

**Description** The **`lk_all_mapped_presentations`** function returns a list of all the presentations that have been defined and mapped in the current application context. See the function **`lk_prologue`** for a definition of the application context.

**Returns** A value of type **`list (integer)`**.

You can cast the value as **`list (Lk_presentation)`**. Each element in the list corresponds to the identifier of a mapped presentation. If there are no mapped presentations, the empty list is returned.

**Example** If you want to unmap all the presentations of the current application context except the one from which the method was called, you can use the **`lk_all_mapped_presentations`** method in the following way:

```
Lk_presentation cp = lk_current_presentation(), p;
o2 list (integer) lp;
...
lp = (o2 list(integer))
lk_all_mapped_presentations();
for (p in lp) {
    if (p!= cp)
      lk_unmap (p);
      ...
}
```

## lk_all_presentations

**Summary** Lists of all the mapped and unmapped presentations defined in current application.

**Syntax** **`lk_all_presentations()`**

**Arguments** None.

**Description** The **`lk_all_presentations`** function returns a list of all presentations that have been defined under the current application context, whether they are mapped or unmapped.

**Returns** A value of type **`list (integer)`**.

You can cast the value as **`list (Lk_presentation)`**. Each element in the list corresponds to the identifier of a mapped presentation. If there are no presentations, the empty list is returned.

**Example**

```
Lk_presentation p;
o2 list (integer) lp;
...
lp = (o2 list(integer))lk_all_presentations();
for (p in lp) {
...
}
...
```

## lk_consult

**Summary** The object or value displayed in a presentation can be consulted.

**Syntax** **lk_consult (***presentation***,** *obj_or_value***)**

**Arguments**

*presentation* The presentation identifier of type **Lk_presentation** is returned by a **present** method or **lk_present** function.

*obj_or_value* The object or value name known to the calling program.

**Description** The **lk_consult** function enables a program to consult the object or value displayed in a presentation.

This value can also be stored in the the program variable that is passed to the function as an argument.

If there are component objects, these are saved in the database even if they are being edited in another presentation: **lk_consult** does not check for consistency here.

This is very useful for data entry routines. You can use it after an **lk_wait** or **lk_grab** function and to return any modifications made in the presentation.

The values from the **lk_consult** function are automatically saved when you click on Pencil and are therefore immediately available to the method or program using the **object->field** construction.

However, **lk_consult** is needed for an unlocked presentation. For example, if you paste an object **obj2** on the presentation of an object **obj1** and click on Pencil, the values of **obj1** are not updated by those of **obj2** because the Pencil click applies only to **obj2.**

A reference to **obj1** shows no change to its value but an **lk_consult** of the presentation shows the values of **obj2**.

Testing and recognizing this difference is the only way a program can know that you have switched objects in a presentation.

### *Important*

The object displayed by the presentation, which was passed to the **lk_present** function as an argument, is always updated by the **lk_consult** method, whatever the value of the *obj_or_value* argument.

**Returns** Zero if successful.

Non-zero if the identifier specified does not exist, i.e. it is not defined by a **present** method or **lk_present** function, or the data types specified in the *presentation* and *obj_or_value* parameters are incompatible.

**Example** The following method checks whether the presentation is modified:

```
o2 Person temp = new Person;
Lk_presentation pr;
temp = self->deep_copy;
pr = lk_present (self, lk_protected(0,0,0));
lk_map (pr, LK_MANUAL, 0,0,0,0);
if (lk_wait(pr) == LK_SAVE) {
   lk_consult (pr, self);
   if ((self->deep_equal (temp)) == true) {...}
   else {...}
}
```

## lk_current_presentation

**Summary** Gives the presentation from which the calling method was invoked interactively.

**Syntax** `lk_current_presentation()`

**Arguments** None

**Description** The **lk_current_presentation** function returns the identifier of the presentation from which the end-user selected a public method from a menu.

The **lk_current_presentation** returns the identifier of the last presentation on the screen for which an object menu was activated. In some cases, this might not be the one from which the calling method was invoked. This is because the calling method might itself create and map presentations, and interactions with these or other presentations might involve the activation of menus.

The only way to be certain that you obtain the presentation from which the calling method was invoked is to call **lk_current_presentation** immediately as the first instruction of the method. This is shown in the example below.

**Returns** A value of type **Lk_presentation** identifying the presentation from whose menu the calling method was invoked. If the calling method was not invoked interactively, or if **lk_current_presentation** was called from an application program or a function, a zero is returned.

**Example** This code erases all presentations of the current application context except the one from which the method was called.

```
Lk_presentation cp = lk_current_presentation(),
p;
o2 list (integer) lp;
...
lp = (o2 list(integer))
lk_all_mapped_presentations();
for (p in lp) {
    if (p!= cp)
      lk_unmap (p);
}
...
```

## lk_delete_all

**Summary** Destroys all existing presentations.

**Syntax** `lk_delete_all()`

**Arguments** None.

**Description** The function **`lk_delete_all`** destroys all existing presentations that have been defined in the current application context.

**Returns** Nothing.

**Example**

```
lk_delete_all();
```

## lk_delete_presentation

**Summary** Removes a presentation.

**Syntax** **lk_delete_presentation (***presentation***)**

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** returned by a **present** method or **lk_present** function.

**Description** The function **lk_delete_presentation** removes a presentation from the system. Any display of the presentation is unmapped and the presentation definition is deleted, reversing the effect of the **present** method or **lk_present** function.

**Returns** If successful the integer zero. Otherwise, it returns a non-zero if the presentation has been locked with an **lk_grab** or **lk_wait** function, and cannot be erased, or the presentation has never been defined with a **present** method or **lk_present** function.

**Example**

```
lk_delete_presentation (my_presentation);
```

## lk_epilogue

**Summary** **lk_epilogue** – Terminates the $O_2$Look application.

**Syntax** **lk_epilogue()**

**Arguments** None.

**Description** The **lk_epilogue** function terminates the $O_2$Look application context initiated by the most recent **lk_prologue** function.

All $O_2$Look windows associated with that context are erased, and the application context that was in force before the most recent **lk_prologue** is reinstated. You can stack and unstack $O_2$Look application contexts in this way.

**Returns** None.

**Example**

```
lk_epilogue();
```

## lk_free

**Summary** Reactivates a suspended method or application program.

**Syntax** **lk_free (***presentation*, *response***)**

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

*response* The integer value returned by the relevant **lk_wait** or **lk_grab** function as its lock is deactivated, can be:

- **LK_SAVE** Proceed as if the user has clicked on the Pencil of the specified presentation.
- **LK_ERASE** Proceed as if the user has clicked on the Eraser of the specified presentation.

**Description** The function **lk_free** reactivates a method or application program suspended by **lk_wait** or **lk_grab**. Not all such locks need an **lk_free** function call; a suspended method or program may also be reactivated by the user clicking on either the Pencil or the Eraser button of the associated presentation. When the **lk_free** call is issued, it specifies the integer value to be returned by the relevant **lk_wait** or **lk_grab** function. Locks must be removed in the reverse order in which they were invoked, i.e., locks are placed on a stack, and must be unstacked in the proper order. This applies regardless of how the lock is terminated (mouse click or **lk_free** function).

**Returns** An integer; zero if all went well, or non-zero if:

- The presentation does not exist or is not mapped.
- The presentation does not currently have a lock to deactivate.
- The lock is not on the top of the lock stack.

**Example**

```
lk_free (my_presentation, LK_ERASE);
```

## lk_grab

**Summary** Suspends execution of the method, program or function.

**Syntax** **lk_grab (***presentation***)**

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

**Description** The function **lk_grab** suspends the method, application program or function in which it occurs. They remain blocked until you click on the Pencil or Eraser button of the specified presentation, or until some other method or application program issues an appropriate **lk_free** call.

All other presentations that have been defined under the current application context are frozen; you cannot manipulate them until you clicks on the Pencil or Eraser button of the **lk_grab** presentation.

A presentation locked with **lk_grab** cannot be erased from another method or program. Except for the fact that **lk_grab** "grabs" the screen, it behaves like **lk_wait**.

**Returns** An integer, which may be one of the following:

- **LK_SAVE**: The user clicked on the Pencil button.
- **LK_ERASE**: The user clicked on the Eraser button.
- A negative integer **LK_FAILED** if the presentation is not currently displayed anywhere, or if it is already locked.

**Example**

```
n = lk_grab (my_presentation);
if (n == LK_SAVE) {
  ...
}
else if (n == LK_ERASE) {
  ...
}
```

## lk_map

**Summary** Displays a presentation defined with the **present** method or **lk_present** function.

**Syntax** **lk_map (***presentation*, *placement*, *link*, *reference*, *x*, *y***)**

**Arguments**

*presentation* The presentation identifier is of type **Lk_presentation** and is returned by a **present** method or **lk_present** function.

*placement* This determines how the presentation is placed on the screen. The placement may also be affected by the *x* and *y* coordinates described below. This parameter is obligatory and has several options:

**LK_MANUAL**

The user clicks any mouse button on the screen at the spot where the upper left corner of the new presentation is to be placed. Before clicking, a "ghost" presentation appears which may be dragged around the screen. If the *x* and *y* parameters are non-zero, the cursor appears offset into the presentation by *x* pixels to the right and *y* pixels down. This is useful for pre-positioning the cursor over a certain field or button.

**LK_MOUSE**

The new presentation appears with its upper left corner where the mouse cursor is; no click is necessary. If the *x* and *y* parameters are non-zero, the cursor appears offset into the presentation by *x* pixels to the right and *y* pixels down. This is useful for pre-positioning the cursor over a certain field or button.

**LK_COORDINATE**

The upper left corner of the new presentation is placed *x* pixels to the right and *y* pixels below the reference point defined in the *reference* parameter. If the *reference* parameter is **LK_SCREEN**, this point is the upper left corner of the screen. If the *reference* parameter is another presentation, then this point is the upper left corner of that presentation.

**LK_RIGHT**

The new presentation is placed *x* pixels to the right of the one defined in the *reference* parameter.

**LK_STACK**

The new presentation is "stacked", partially covering the one defined in the *reference* parameter, but offset *x* pixels to the right and *y* pixels down.

*link* This argument specifies how the new presentation is linked to the one specified by *reference*. This parameter is not required (i.e. may be left zero) if there is no link.

The options are:

**LK_FREE**

No link at all, apart from the initial placement (specified in the *placement* parameter). When the *reference* presentation or any presentation linked to it is moved, resized, saved, erased or iconified, the action does not affect any other presentation.

**LK_GEOMETRIC**

When the *reference* presentation or any presentation linked to it is moved or resized, the other linked presentations reposition themselves to keep the same relative positions. When any of the linked presentations is iconified, the other ones also disappear into the same icon, and reappear together when it is de-iconified. However, save and erase operations are not propagated from one linked presentation to another.

**LK_CONTENT**

When the *reference* presentation or any presentation linked to it is unlocked, unmapped or saved onto the database, the same operation occurs in all the other linked presentations. However, move, resize and iconification operations on one presentation have no effect on any linked ones.

**LK_SLAVE**

All move, resize, iconification, unlock, unmap and save operations on one linked presentation are propagated to all the other linked presentations.

*reference* The identifier of another presentation, already displayed, next to which the current one is to be placed, and to which the current one is to be linked. For placement, the upper left corner of the *reference* presentation is the reference point. If the current presentation is not to be linked to any existing one, then the argument **LK_SCREEN** may be used, or the parameter may be left zero; in this case, the reference point is the top left corner of the screen.

*x* — The number of horizontal pixels separating the upper left corner of the new presentation from the reference point defined in the *reference* parameter. This is required for the **LK_COORDINATE**, **LK_LEFT**, **LK_RIGHT** and **LK_STACK** placements; it is optional for **LK_MANUAL** and **LK_MOUSE**; in other cases it is ignored. If the global graphic resource **presSpacing** is defined, the horizontal displacement is *x* + **presSpacing**.

*y* — The number of vertical pixels separating the upper left corner of the new presentation from the reference point defined in the *reference* parameter. This is required for the **LK_COORDINATE**, **LK_TOP**, **LK_BOTTOM** and **LK_STACK** placements; it is optional for **LK_MANUAL** and **LK_MOUSE**; in other cases it is ignored. If the global graphic resource **presSpacing** is defined, the vertical displacement is *y* + **presSpacing**.

**Description** The function **lk_map** displays a presentation which has been defined with the **present** method or **lk_present** function. Its parameters identify the presentation to display and specify where to display it.

**Returns** An integer: zero if all went well, or non-zero if:

- The presentation specified in the *presentation* parameter is already displayed.
- The presentation specified in the *reference* parameter is not yet displayed.
- The identifier specified in either the *presentation* parameter or the *reference* parameter does not exist; that is, it has not been defined by a **present** method or **lk_present** function.

**Example**

```
lk_map (p1, LK_COORDINATE, LK_FREE, LK_SCREEN, 50, 10);
lk_map (p2, LK_COORDINATE, LK_GEOMETRIC, p1, 50, 30);
lk_map (p3, LK_MANUAL, 0, 0, 0, 0);
lk_map (p4, LK_STACK, LK_SLAVE, p2, 30, 30);
lk_map (p5, LK_LEFT, LK_FREE, p4, 20, 0);
```

## lk_present

**Summary** Defines a presentation for an $O_2$ object or value.

**Syntax** **lk_present (***obj_or_value***,** *mask* **[,** *ed_name***,** *rcount***,** *resources***])**

**Arguments**

*obj_or_value* The object or value name as known to the calling program.

*mask* The mask (of type **Lk_mask**) you want to use to in the presentation. If left zero, the method takes the **lk_generic** default mask. If the mask you choose is inconsistent with the value or object you want to display, you get a warning and the **lk_generic** mask is automatically used.

*ed_name* A presentation identifier which you can then use to specify its graphic resources in a resource file. The *ed_name* default value is **GenPresentation**.

*rcount* The number of resources in the *resources* argument.

*resources* A pointer to an array of graphic resources to be customized. Each resource is of type **Lk_resource**.

**Description** The function **lk_present** defines a presentation for an $O_2$ object or value. It returns a presentation identifier to the calling program. The presentation mask specifies how data is displayed in the Motif window. For an object, the presentation also includes the menu of public methods attached to the object.

**Returns** The identifier (of type **Lk_presentation**) representing the presentation built. This value is used in all other function calls manipulating the presentation.

**Example**

```
Lk_presentation p1, p2;
Lk_resource v2res[4];
...
p1 = lk_present (object1, lk_protected());
p2 = lk_present (value2, 0, "value2", 4, v2res);
```

**Resources** Section 4.6 lists the graphic resources which can be customized for an individual presentation.

## lk_prologue

**Summary** Initializes an application context.

**Syntax** **lk_prologue (***host*, *context_name*, *context_class*, *rcount*, *resources***)**

**Arguments**

*host* The name of the workstation (string) on which the $O_2$Look presentations appear. The Motif window manager must be running on that machine. If *host* is zero, the **-display** option value is used if one exists. If **-display** does not exist, the value of the Unix environment variable **DISPLAY** is used.

*context_name* The application context name (string). This name appears as the first name in an $O_2$Look resource string used in a resource file to specify default graphic resources for an application context. The *context_name* is also the default window title that appears in the title bar of all presentations defined under the application context. The default is the Unix command (usually the string **"o2"**) used to initiate the $O_2$ session. This is also the default context name between the time that $O_2$ is launched and the first **lk_prologue** function executed. Once an application is executed, the default context name is taken from the name of that application.

*context_class* A string representing the name of the context class of the application. By default (that is, if the parameter is zero), the class name is simply the *context_name* argument with its first letter converted to upper-case. Like *context_name*, *context_class* may be used as the first part in an $O_2$Look resource string in a resource file. However, instead of specifying graphic resources for the application being run, it specifies them for all applications of this class.

*rcount*: An integer count of the number of special resources given in the *resources* argument.

*resources* A pointer to an array of graphic resources to be customized in the application context. Each resource in the array is of type **Lk_resource**. Only global resources are specified here.

**Description** The **lk_prologue** function initializes an application context. The currently active application context is superseded by a new one.

One or more resource files containing default graphic resource specifications are read and implemented. The name and location of these files can vary depending on the $O_2$ installation directory, the current value of the Unix environment variables **O2RESOURCE** and **HOME** (the user home directory), and the *context_name*.

$O_2$Look searches for the resource files in the following order:

1. If **O2RESOURCE** is defined, $O_2$Look attempts to load the file
   *O2_installation_directory*/**look/$O2RESOURCE/***context_name*
   If **O2RESOURCE** is not defined, $O_2$Look tries
   *O2_installation_directory*/**look/***context_name*

2. $O_2$Look then looks for
   **$HOME/.LKdefaults**

If neither 1 nor 2 exists, no resource file is loaded. If both exist, they are both loaded, and any conflicting resource definitions are resolved by giving **.LKdefaults** priority.

The new application context remains in force until either a new **lk_prologue** function replaces it or an **lk_epilogue** function terminates it and restores the previous context. Application contexts can be stacked and unstacked to any reasonable depth in this way. Note that application contexts and graphic resource definitions are not cumulative: one context completely replaces another. An application context defines the resource string prefixes used when customizing presentation windows and display masks. It also defines the boundaries beyond which an **erase_all** method or an **lk_delete_all** function will not go.

An $O_2$ application generally begins with **lk_prologue** and ends with **lk_epilogue**, and corresponds to one end-user task. In practice, your end-user may wish to interrupt one application and run another one. A programmer, for example, may want to interrupt an application they are writing, test a second one, and go back to the first afterward. This is why application contexts can be stacked. When an $O_2$ session starts up, the automatically system initializes a default application context called **o2** unless the **-alpha** command is used, in which case no application context is established and no $O_2$Look functions operate.

**Returns** Nothing.

**Resources** Section 4.5 lists the graphic resources which can be customized for an application context

**Example**

```
lk_prologue (0, "tour_booking", "TravelAgency", 0, 0);
```

## lk_refresh_presentation

**Summary** Updates a presentation.

**Syntax** **lk_refresh_presentation (***presentation*, *obj_or_value*, *mask* *[*, *ed_name*, *rcount*, *resources]***)**

**Arguments**

*presentation* The presentation identifier of type **Lk_presentation** returned by a **present** method or **lk_present** function.

*obj_or_value* The name of the object or value (as known to the calling program) whose database value is used to refresh the specified presentation. This is usually, but not necessarily, the same object or value that was used to construct the presentation in the first place; in any event, it must be type-compatible with the last object or value displayed in the presentation.

*mask* The display mask (a value of type **Lk_mask**) to be used in the redefinition of the presentation. If left zero, the system retains the mask that was specified in the original **lk_present** function and must not have been destroyed by **lk_destroy_mask**.

*ed_name* Not yet in use.

*rcount* An integer count of the number of special resources given in the *resources* argument.

*resources* A pointer to an array of graphic resources to be customized in the presentation (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**.

**Description** The function **lk_refresh_presentation** brings a presentation of an object or a value up to date. If it has been modified on the database (by any other user or in any other presentation) since the specified presentation was defined, the presentation is updated with the new values.

If any components of the object or value are themselves objects, these are also refreshed. For mapped presentations, the display is redrawn to show the new values (and any unsaved changes are discarded).

This function is useful for maintaining consistency between the data on the database and that data displayed on the screen, or for abandoning any screen changes and reverting to the database value of an object or value.

Refreshing a presentation of an object involves relaunching the **title** and **bitmap** methods for that object, but not the **menu** method. The **lk_refresh_presentation** function may also be used to alter the display mask, the editor name, or most of the graphic resources that were originally specified in the **lk_present** function for the presentation.

Some graphic resources, however, are unaffected by the refreshing process. Note also that this function does not reread the graphic resource file; any resource changes must be specified in the *resources* parameter.

**Returns**

An integer; zero if all went well, or non-zero if:

- The identifier specified in the *presentation* parameter does not exist; that is, it has not been defined by a *present* method or **lk_present** function.
- The data types of the presentation specified in the **presentation** parameter and of the object or value specified in the *obj_or_value* parameter are not compatible.

**Resources**

Section 4.6 lists the graphic resources which may be customized for an individual presentations.

**Example**

This code redraws the presentation **pr1** based on the current contents of object **client**.

```
lk_refresh_presentation (pr1, client, 0);
```

## lk_unmap

**Summary** Erases the presentation.

**Syntax** **lk_unmap (***presentation***)**

**Arguments**

| | |
|---|---|
| *presentation* | The presentation identifier of type **Lk_presentation** and returned by a **present** method or **lk_present** function. |

**Description** The function **lk_unmap** erases a presentation from the screen thereby reversing the **lk_map** function. It leaves the definition of the presentation intact, so that future **lk_map** instructions can re-display the presentation.

**Returns** An integer; zero if all went well, or non-zero if:

- The presentation has never been defined with a **present** method or **lk_present** function.
- The presentation is not currently displayed anywhere.
- The presentation has been locked with an **lk_wait** or **lk_grab** function, and cannot be unmapped.

**Example**

```
lk_unmap (my_presentation);
```

## lk_wait

**Summary** Suspends the running of a method, program, or function.

**Syntax** **lk_wait** (*presentation*)

**Arguments**

*presentation* The identifier of the presentation to be locked. This is the value, of type **Lk_presentation**, returned by a **present** method or **lk_present** function.

**Returns** An integer, which may be one of the following:

- **LK_SAVE**: The user clicked on the Pencil button.
- **LK_ERASE**: The user clicked on the Eraser button.
- A negative integer **LK_FAILED** if the presentation is not currently displayed anywhere, or if it is already locked.

**Description** The function **lk_wait** suspends the method, application program or function in which it occurs. This remains blocked until you click on the Pencil or Eraser button of the specified presentation, or until some other method or application program issues an appropriate **lk_free** call.

You can, however, continue to manipulate any other presentation on screen. Using an **lk_wait** is the normal way in which a method or application program pauses to accept user input.

While an **lk_wait** is in effect, the system continues to process all events coming from the end-user, for all presentations on the screen. You can modify the contents of other presentations, or copy or cut-and-paste whole objects or values, or trigger methods.

By clicking on the Pencil button of any presentation other than a locked one, you ensure that changes made to that presentation are recorded on the database. By clicking on the Eraser button of any presentation other than a locked one, you abandon the changes and the presentation is erased from the screen.

The running of the suspended method or program only resumes when you click on the Pencil or Eraser button of the presentation affected by **lk_wait**.

Changes made to this presentation are not recorded on the database if the Pencil button is clicked, and the presentation is not destroyed if the Eraser button is clicked. Information as to how the lock was terminated is returned to the method or application program which issued the **lk_wait** call.

It is up to the method or program to handle the situation in the appropriate way: to invoke the proper **lk_consult** or **lk_delete_presentation** functions, to test any new values entered by the end-user, and to store them on the database if necessary. A presentation locked with **lk_wait** cannot be erased from another method or program. The **lk_wait** function returns an integer which, if not negative (i.e. if not an error code), indicates the way in which the **lk_wait** was terminated:

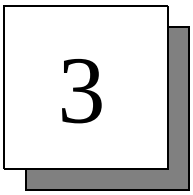
- If you click on the Pencil button, the value **LK_SAVE** is returned.
- If you click on the Eraser button, the value **LK_ERASE** is returned.
- If an **lk_free** function was executed, the value of the *response* parameter of that function is returned.

The **lk_wait** function places a lock on top of a "stack" of locks in effect for the current application. An application may have any number of locks in effect at once, but the locks must be terminated in the reverse order in which they were invoked. If, for example, you click on the Pencil button of a locked presentation, but this presentation was not the most recent one locked, an appropriate error message appears.

Methods and programs using the **lk_free** function must also be careful to undo locks in the proper order. When a presentation is unlocked, any other presentations linked to it by a **LK_CONTENT** or **LK_SLAVE** link are also unlocked. Apart from respecting the order of the lock stack, an **lk_wait** lock on one presentation does not affect the execution of other methods in an application.

**Example**

```
n = lk_wait (my_presentation);
if (n == LK_SAVE) {
  ...
}
else if (n == LK_ERASE) {
  ...
}
```



# Masks

A mask definition describes a template for the format of a presentation. For example, which tuple attributes are to be displayed and which ones can be modified. You can also use masks to specify the graphic resources of your presentations.

This chapter introduces these display masks and describes how to use them. It then gives the exact syntax of all the mask functions.

It is divided into the following sections:

- Introduction
- Functions

# 3 Masks

## 3.1 Introduction

$O_2$Look masks are templates for the display and editing of values.

In most cases, when an object or value is displayed under $O_2$Look, no mask need be specified.

The system uses a default mask derived from the type specification of the object or value concerned.

However, under special circumstances, you may wish to design your own display mask, supplementing or even violating the encapsulation rules specified in the class description, or altering the graphic resources used to display data.

The masks described in this chapter enable you to now do this.

### Mask Functions

Each $O_2$Look mask function described in Section 3.2 creates and returns a mask.

There is a generic display mask for every data type defined in an $O_2$ system:

- ***lk_atomic***

  To create a mask for atomic values: integers, real numbers, characters, character strings and booleans,

- ***lk_set***

  To create a mask for sets and unique set values

- ***lk_list***

  To create a mask for list values.

- ***lk_tuple***

  To create a mask for tuple values

  There are also various generic mask functions:

- ***lk_generic***

  To create a mask according to the type specification. This is the default mask for $O_2$ values.

- ***lk_protected***

  To create a mask according to the class specification.

- ***lk_object***

  To create a mask for an object.

- ***lk_method***

  To create a mask according to the method specification.

  There are also special displays specific editors:

- ***lk_specific***

  To create a mask for a specific editor: bitmap, date, dialog box, graph, image, hyper and text.

A display mask associated with a complex object or value is composed of several subsidiary masks.

This display masks may be used in the **lk_present** and **lk_refresh_presentation** functions in place of the two default masks. Often, one mask will include one or more sub-mask specifications.

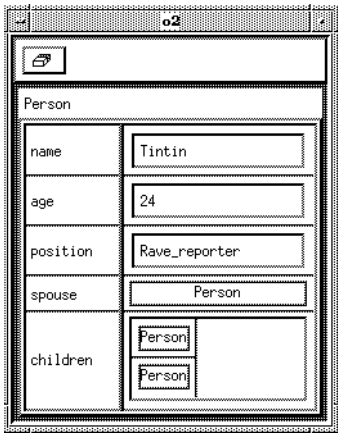
This is the case, for example, with the mask that displays a set value, created by the **lk_set** mask function**.** **lk_set** requires the specification of a sub-mask of the appropriate type for the elements of the set. The elements might even be tuples, or tuple-structured objects, which might contain a list as an attribute, and so on.

Masks and sub-masks may be nested to any reasonable depth.

The tuple mask **lk_tuple**, for example, can contain any or all of the other types masks. Such complex masks may be nested to any level.

This is illustrated in Figure 3.1 where the **lk_tuple** mask contains three **lk_atom** masks, a **lk_generic** mask and a **lk_list** mask which itself contains a **lk_generic** mask.

# 3 Masks



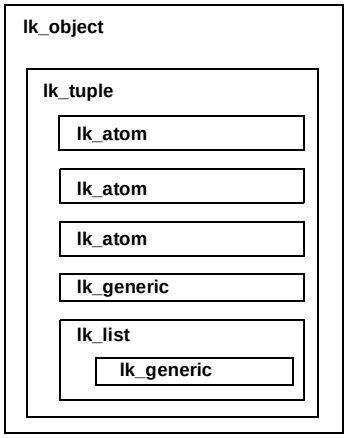




*Figure 3.1: Presentation and mask*

All masks support standard operations such as keyboard input, copying, cut-and-paste and erasing, although the you may suppress these operations for certain masks. Some of these operations may appear in a special menu related to the mask itself.

**Note**

There is also a **`lk_destroy_mask`** function that deletes any mask created in any of the previous functions.

## Resource specification in Functions

$O_2$Look presentations are simply Motif windows, and as such their appearance and behavior may be customized like any other Motif window, by specifying graphic *resources*. A resource is any customizable aspect of the X Window System.

The X Window System maintains a database of currently-active resources, and $O_2$Look provides a means of altering the contents of that database as they apply to $O_2$Look presentations.

In this way, default application resources may be established; multilingual or other situation-specific application appearance and behavior may be programmed; and the end user may customize applications to his/ her particular taste.

Resources can be specified in either (or both) of two ways: through a *`resource file`* containing resource commands, or through the passing of a list of resource values in an $O_2$Look function call.

Whenever an $O_2$Look resource is specified with two conflicting values, one from a resource file and the other from a function call, the value specified in the function call takes precedence.

Most of these mask functions include an option for specifying graphic resource arrays.

A resource list involves a count of the resources in the array (*rcount*) followed by a pointer to the array itself (*resources*); the elements of the array are each of type **Lk_resource**.

Graphic resources may also be specified for individual masks in a default resource file. The mask is known by its editor name, which you specify in the *ed_name* argument of the mask function.

Resources that may be specified in these ways include color, typeface, and protection (that is, whether or not the user is to be allowed to change the displayed data). In the case of structured values (and, therefore, complex masks), the protection of the structure (tuple, set, list) need not be the same as that of the elements. The structure's protection refers to the ability to create new elements, or delete old ones, whereas the ability to view or change the value of the elements is governed by the protection defined for each element.

This may be done either when a presentation or mask is created (by passing a list of resources to the function that creates it), or later (through the **lk_refresh_presentation** function). In either case, the caller must supply two Arguments: a count of the number of resources to be customized (generally called *rcount*) and a pointer to an array of resource specifications (usually called *resources*).

Each element in the array must be of type **Lk_resource**, which is known to the system as the following C structure:

```
typedef struct {
   char *name;
   char *value;
} Lk_resource;
```

The **name** string identifies the resource, and the **value** string indicates what it is to be set to.

Note that the **name** string is not the full name specification required for a resource file; it is only the last part of that specification, such as **"fontList"** or **"foregroundColor"**.

This is because the function itself already knows which presentation or mask is to be affected by the change.

The $O_2$Look functions that accept resource arrays are:

- ***lk_prologue***

  resource specifications affect all presentations under the newly-initiated application context. Only global resources may be specified here.

- ***lk_present***

  resource specifications affect only the presentation being created.

- ***lk_refresh_presentation***

  resource specifications affect only the presentation being refreshed. The resources are the same as for **`lk_present`**.

- ***All mask functions***

  resource specifications affect only the mask being created. However, this mask may be used as a sub-mask in several presentations at once. The resources that may be customized vary from one type of mask to another.

## 3.2 Functions

This section details the following mask functions:

- **lk_atom**
- **lk_destroy_mask**
- **lk_generic**
- **lk_list**
- **lk_method**
- **lk_object**
- **lk_protected**
- **lk_set**
- **lk_specific**
- **lk_tuple**

# 3 Masks

## lk_atom

**Summary** Creates a mask for an atomic value.

**Syntax** **lk_atom (***ed_name*, *rcount*, *resources***)**

**Arguments**

| | |
|---|---|
| *ed_name* | A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask. |
| *rcount* | An integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**. |

**Description** The **lk_atom** mask function creates a mask for an atomic value; there is no need to specify which kind of atomic value is involved. This function can be used directly in an **lk_present** or **lk_refresh_presentation** function to present atomic values or as an element mask specification for the following functions:

- **lk_object** for an atomic-valued object
- **lk_tuple** for an atomic attribute of a tuple
- **lk_set** for a set of atomic values
- **lk_list** for a list of atomic values

**Returns** A mask identifier of type **Lk_mask**.

**Example**

```
o2 string name;
Lk_presentation p;
name = "Caroline";
p = lk_present (name, lk_atom(0, 0, 0), 0, 0, 0);
lk_map (p, LK_MANUAL, 0, 0, 0, 0);
lk_grab(p);
```

You get the following window:

*Figure 3.2: Using lk_atom mask*

## lk_destroy_mask

**Summary** Deletes a mask created by any of the previous functions.

**Syntax** **lk_destroy_mask (***mask***)**

**Arguments**

| | |
|---|---|
| *mask* | the mask identifier (of type **Lk_mask**) of the mask to be destroyed. |

**Description** The **lk_destroy_mask** function deletes a mask created by one of the mask creation functions. A reference to a mask should not be made after its deletion. Additionally, if the memory resource is at a premium, we recommend you use this function to recover the space used by the mask.

The system does not verify whether a mask is in use before destroying it. In particular, all methods and functions that refresh a presentation will expect that the mask used for that presentation still exists. If it does not, a run-time exception will occur.

**Returns** Nothing

**Example**

```
Lk_presentation p;
Lk_mask m = lk_generic();
o2 Thing x;
...

p = lk_present (x, m);
lk_destroy_mask (m);
lk_map (p, LK_MOUSE, LK_FREE, LK_SCREEN, 0, 0);
...

lk_delete_presentation (p);
```

The mask **m** may be destroyed immediately after being used by **lk_present** to define the presentation. However, any subsequent **refresh_all** method or **lk_refresh_presentation** function would fail (except in the case where **lk_refresh_presentation** provides a new mask).

## lk_generic

**Summary** Creates a mask according to the type specification.

**Syntax** **`lk_generic()`**

**Arguments** None.

**Description** The **`lk_generic`** mask function creates a mask according to the type specification of the value or object concerned.

This is the default mask for $O_2$ values. When applied to an object, this mask produces an iconic representation of the object.

This function is used principally as an element mask specification for the following functions:

- **`lk_object`** for an iconic representation of the object
- **`lk_tuple`** for any attribute of a tuple
- **`lk_set`** for any set
- **`lk_list`** for any list

It may also be used directly in an **`lk_present`** or **`lk_refresh_presentation`** function.

Default names are used in the created mask according to the type specification shown in Table 3.1 :

Table 3.1

| Type | Default name |
|---|---|
| char | GenChar |
| integer | GenInteger |
| real | GenReal |
| boolean | GenBoolean |
| string | GenString |
| byte | GenByte |
| list | GenList |
| uset | GenSet |
| set | GenBag |
| object | GenObject |

**Returns** A mask identifier of type **`Lk_mask`**.

**Example**

```
o2 tuple (name: string, age: integer) a;
Lk_presentation p;
a.name = "George";
a.age = 45;
p = lk_present (a, lk_generic());
lk_map (p, LK_MANUAL, 0, 0, 0, 0);
lk_grab(p);
```

You get the following window:

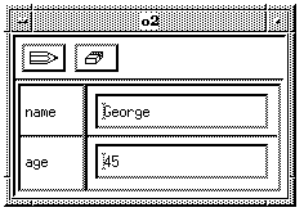



*Figure 3.3: Using the generic mask*

## lk_list

**Summary** Creates a mask for a list value.

**Syntax** **lk_list (***ed_name*, *rcount*, *resources*, *element_mask***)**

**Arguments**

| | |
|---|---|
| *ed_name* | A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask. |
| *rcount* | An integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**. |
| *element_mask* | A mask identifier (of type **Lk_mask**) appropriate to the data type of the elements of the list. |

**Description** The **lk_list** mask function creates a mask for a list value.

This function is used principally as an element mask specification for the following functions:

- **lk_object** for a list-valued object
- **lk_tuple** for a list-structured attribute of a tuple
- **lk_set** for a set of lists
- **lk_list** for a list of lists

It may also be used directly in an **lk_present** or **lk_refresh_presentation** function, for a presentation of a list-structured value.

**Returns** A mask identifier of type **Lk_mask**.

**Example** A generic mask for a list of integers would be specified as:

```
run body{
o2 list (integer) numbers;

Lk_presentation p;
Lk_mask mk;

numbers = list (99, 76, 56);
mk = lk_list ("foo", 0, 0, lk_atom(0, 0, 0));
p = lk_present (numbers, mk, 0, 0, 0);
lk_map (p, LK_MANUAL, 0, 0, 0, 0);
lk_grab(p);
}
```

This gives the following window:

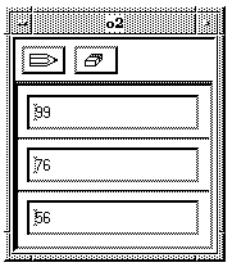



*Figure 3.4: Using the lk_list mask*

## lk_method

**Summary** Creates a mask according to the execution of a specified method.

**Syntax** **lk_method (***method_name***)**

**Arguments**

| | |
|---|---|
| *method_name* | the method defined for the class concerned. This method takes no parameters and returns a mask identifier. |

**Description** The **lk_method** mask function creates a mask according to the method specified.

As it specifies a method that is passed to an object and which builds up the object mask, **lk_method** cannot be applied to values.

You invoke the method when the presentation is defined (i.e, **lk_present**).

You can use different graphic resources according to the attribute's current value.

Redefine the called method in different subclasses to produce different masks according to the class membership of the various elements of a heterogeneous set or list. You can use this function anywhere **lk_protected** or **lk_object** is used.

**Returns** A mask identifier of type **Lk_mask**.

**Example** The following example shows how to use the **lk_method** method

The example shown below takes the **balance** attribute of an object of class **Client** is displayed in **red** if the balance is negative, and in the **normal** color if it is zero or positive.

```
method body redmask: integer in class Client {
Lk_mask m;
Lk_attribute_mask tmasks[3];
Lk_resource redres;
redres.name = "foregroundColor";
redres.value = "red";
tmasks[0].name = "name";
tmasks[0].mask = lk_atom (0, 0, 0);
tmasks[1].name = "client_number";
tmasks[1].mask = lk_atom (0, 0, 0);
tmasks[2].name = "balance";
if (self->balance < 0)
  tmasks[2].mask = lk_atom (0, 1, &redres);
else
  tmasks[2].mask = lk_atom (0, 0, 0);
m = lk_object (0, 0, 0, lk_tuple (0, 0, 0, 3, tmasks));
return m;
}
```

You use the mask to build a presentation in an application program:

```
o2 Person p;
Lk_presentation i;
...
i = lk_present (p, lk_method ("redmask"));
```

## lk_object

**Summary** Creates a mask for an object.

**Syntax** **lk_object (***ed_name*, *rcount*, *resources*, *value_mask***)**

**Arguments**

| | |
|---|---|
| *ed_name* | A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask. |
| *rcount* | An integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**. |
| *value_mask* | A mask identifier (of type **Lk_mask**) for the value encapsulated in the object (that is, not another **lk_object** and not **lk_protected** or **lk_method**). If *value_mask* is zero, an icon display is generated for the object. |

**Description** The **lk_object** mask function creates a mask for an object. An **lk_object** mask resembles the mask for the value encapsulated by the object.

The main difference is that an object mask also provides for a menu of public methods attached to the object.

The function may be used with a **present** method or **lk_present** or **lk_refresh_presentation** function, or as an element mask specification for the following functions:

- **lk_tuple** for an object attribute of a tuple
- **lk_set** for a set of objects
- **lk_list** for a list of objects

**Returns** A mask identifier of type **Lk_mask**.

**Example** In the parameters of the object mask, you give the mask that corresponds to its value.

```
run body{
Lk_mask mk;
Lk_presentation pid;
o2 Person e = new Person;
e->name.last_name ="Swanson";
e->name.first_name="George";

e->birthdate= new Date (30, 06, 61);
e->address= new Address;
e->address->city = "Tuscon";
e->address->country =  "USA";

mk = lk_object(0, 0, 0, lk_generic());
pid = lk_present (e, mk);

lk_map (pid, LK_MOUSE, 0, 0, 0, 0);
lk_grab (pid);
}
```

You get the following window:

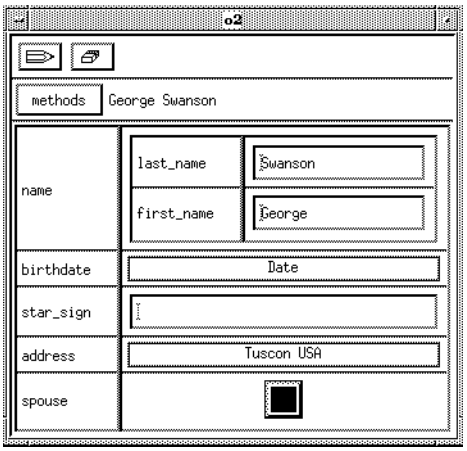



*Figure 3.5: Using the lk_object mask*

**lk_object** shows all the attributes even if they are private or read-only. See Figure 3.6 which shows how to not display private or read-only attributes using **lk_protected**.

## lk_protected

**Summary** Creates a mask according to the class specification.

**Syntax** **lk_protected (***ed_name***,** *rcount***,** *resources***)**

**Arguments**

| | |
|---|---|
| *ed_name* | A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask. |
| *rcount* | An integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**. |

**Description** The **lk_protected** mask function creates a mask according to the class specification of an object. This mask therefore cannot be applied to values.

### Note

Unlike **lk_generic** and **lk_object**, the **lk_protected** mask takes into account the encapsulation specified in the class definition of the object being displayed.

Private tuple attributes are not displayed, and read-only attributes may not be edited. If the entire type structure of the object is private, the object is represented in iconic form only.

This function may be used with a **present** method or **lk_present** or **lk_refresh_presentation** function, or as an element mask specification for the following functions: **lk_tuple** (for an object attribute of a tuple); **lk_set** (for a set of objects); and **lk_list** (for a list of objects).

**Returns** A mask identifier of type **Lk_mask**.

**Example** This uses the same example as for **lk_object** above.

```
run body{
Lk_mask mk;
Lk_presentation pid;
o2 Person e = new Person;
e->name.last_name ="Swanson";
e->name.first_name="George";

e->birthdate= new Date (30, 06, 61);
e->address= new Address;
e->address->city = "Tuscon";
e->address->country =  "USA";


mk = lk_protected(0, 0, 0);
pid = lk_present (e, mk);

lk_map (pid, LK_MOUSE, 0, 0, 0, 0);
lk_grab (pid);
}
```

You get the following window:

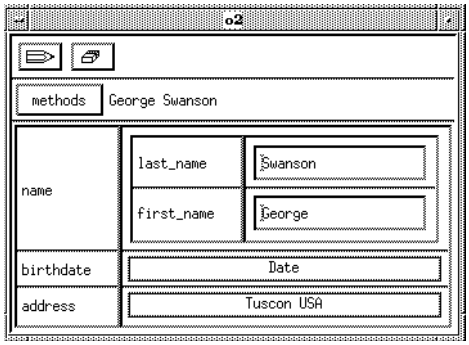



*Figure 3.6: Using the lk_protected mask*

**lk_protected** respects the class encapsulation and only shows public attributes. See Figure 3.5 which shows how to display private or read-only attributes as well as public attriubtes using **lk_object**.

# 3 Masks

## lk_set

**Summary** Creates a mask for a set value.

**Syntax** **lk_set (***ed_name*, *rcount*, *resources*, *element_mask***)**

**Arguments**

| | |
|---|---|
| *ed_name* | A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask. |
| *rcount* | An integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**. |
| *element_mask* | A mask identifier (of type **Lk_mask**) appropriate to the data type of the elements of the set. |

**Description** The **lk_set** mask function creates a mask for a set value (whether a unique set or an ordinary set).

This function is used principally as an element mask specification for the following functions:

- **lk_object** for a set-valued object
- **lk_tuple** for a set-structured attribute of a tuple
- **lk_set** for a set of sets
- **lk_list** for a list of sets

It may also be used directly in an **lk_present** or **lk_refresh_presentation**function, for a presentation of a set-structured value.

**Returns** A mask identifier of type **Lk_mask**.

**Example** A generic mask for a set of strings would be specified as:

```
run body{
o2 set (string) quotes;

Lk_presentation p;
Lk_mask mk;

quotes= set ("To be or not to be",
              "He who laughs last",
             "Look before your leap");
mk = lk_set ("foo", 0, 0, lk_atom(0, 0,
0));
p = lk_present (quotes, mk, 0, 0, 0);
lk_map (p, LK_MANUAL, 0, 0, 0, 0);
lk_grab(p);
}
```

This gives the following window

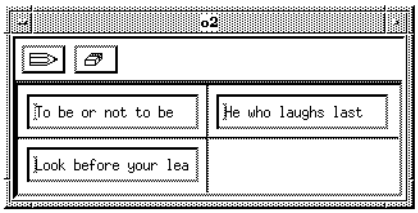



*Figure 3.7: Using the lk_set mask*

## lk_specific

**Summary** Creates a mask for a specific editor.

**Syntax** **lk_specific (***ed_name, rcount, resources, specific_editor, mcount, masks***)**

**Arguments**

*ed_name* A string representing the name of the mask. This name may be used in a resource file to specify graphic resources for this individual mask.

*rcount* An integer count of the number of special resources given in the *resources* argument.

*resources* A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**.

*specific_editor*

A string representing the name of the specific editor. The specific editors available are described below.

*mcount* The number of sub-masks required by the specific editor. This number is given for each specific editor described below.

*masks* A pointer to a table of mask identifiers (each of type **Lk_mask**). This is required, of course, only if *mcount* is greater than zero.

**Description** The **lk_specific** mask function creates a mask for a specific editor. The name of the specific editor is passed as an argument, along with any sub-masks that the specific editor may require. The possibilities are described below.

**Returns** A mask identifier of type **Lk_mask**.

**Example** A specific editor is a mask for a specific type of object or value. The following specific editors are available:

**bitmap**, **date**, **dialog**, **graph**, **hyper**, **image** and **text**

For a full description on how to use these specific editors and their graphic resources, please refer to the $O_2$Kit Reference manual. For the specific graph editor, refer to the $O_2$Graph Reference manual.

## lk_tuple

**Summary** Creates a mask for a tuple value.

**Syntax** **lk_tuple (***ed_name, rcount, resources, acount, apointer***)**

**Arguments**

*ed_name* The mask name which can be used in a resource file to specify graphic resources for the individual mask.

*rcount* The number in integers of special resources given in the *resources* argument.

*resources* A pointer to an array of graphic resources to be customized in the mask (unless *rcount* is zero). Each resource in the array is of type **Lk_resource**.

*acount* The number of tuple attributes to be displayed. If zero, all attributes are displayed, each with its default mask.

*apointer* A pointer to an array of attribute mask specifications. Each array element corresponds to one displayable tuple attribute which is an instance of a C structure:

```
typedef struct {
   char *name;
   Lk_mask mask;
} Lk_attribute_mask;
```

**name** is the tuple attribute name, as known to the calling program**. mask** is a mask identifier (of type **Lk_mask**) corresponding to the attribute type. If a **name** is wrong (i.e., if it does not correspond to an attribute), you get a warning message and no attribute is displayed for that nam*e.* If all of the names are wrong, all attributes are displayed, as if **acount** is zero. Not every tuple attribute need be included in the array, and those that are included may appear in any order. Absent attributes are treated as fully encapsulated and are not included in the tuple mask ( *acount* may be less than the number of tuple attributes defined in the schema). The order of the array elements determines the order of the attributes displayed in the mask.

One **lk_tuple** mask may contain another as a **Lk_mask** specification; these may be nested to any reasonable depth.

**Description** The **lk_tuple** mask function creates a mask for a tuple value.

This function is used principally as an element mask specification for the following functions:

- **lk_object** for a tuple-valued object
- **lk_tuple** for a tuple-structured attribute of a tuple
- **lk_set** for a set of tuples
- **lk_list** for a list of tuples

It may also be used directly in an **lk_present** or **lk_refresh_presentation** function, for a presentation of a tuple-structured value.

**Returns** A mask identifier of type **Lk_mask**.

**Example** In the following example uses the class Person which has several tuple attributes and uses the **lk_tuple** mask to display only the person's last name and star sign:

```
run body{
Lk_mask mk;
Lk_presentation pid;
o2 Person e = new Person;
Lk_attribute_mask pers_tup_mask[2];
pers_tup_mask[0].name = "last_name";
pers_tup_mask[0].mask = lk_generic();
pers_tup_mask[1].name = "star_sign";
pers_tup_mask[1].mask = lk_atom(0, 0, 0);

e->last_name ="Swanson";
e->star_sign="Cancer";

mk = lk_object(0, 0, 0,
               lk_tuple (0, 0, 0, 2, pers_tup_mask));

pid = lk_present (e, mk, 0, 0, 0);
lk_map (pid, LK_COORDINATE, LK_FREE, LK_SCREEN, 50, 10);
lk_grab(pid);
}
```

This code gives the following window:

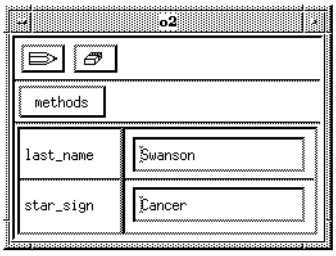



*Figure 3.8: Using the lk-tuple mask*

# 3 Masks

# 4 Graphic Resources

This chapter gives all the various graphic resources for $O_2$Look and is divided into the following sections:


- Using graphic resources
- Resource files
- Resource arrays
- How to read resource tables
- Global resources
- Presentation resources
- Graphic resources for integers
- Graphic resources for real numbers
- Graphic resources for characters
- Graphic resources for strings
- Graphic resources for boolean values
- Graphic resources for lists
- Graphic Resources for sets
- Graphic resources for tuples

# 4 Graphic Resources

## 4.1 Using graphic resources

$O_2$Look presentations are simply Motif windows, and as such their appearance and behavior may be customized like any other Motif window, by specifying graphic resources. A resource is any aspect of the X Window System that can be customized.

The X Window System maintains a database of currently-active resources, and $O_2$Look provides a means of altering the contents of that database as they apply to $O_2$Look presentations.

In this way, default application resources may be established; multilingual or other situation-specific application appearance and behavior may be programmed; and the end user may customize applications to his/ her particular taste.

Resources can be specified in either (or both) of two ways: through a resource file containing resource commands, or through the passing of a list of resource values in an $O_2$Look function call.

Whenever an $O_2$Look resource is specified with two conflicting values, one from a resource file and the other from a function call, the value specified in the function call takes precedence.

## 4.2 Resource files

$O_2$Look-related resources may be specified in one or more resource files which $O_2$Look reads as it starts up or when it executes an **`lk_prologue`**.

The names of the files read are determined by several variables: the name of the $O_2$ installation directory, as established at installation time; the values of the user's **`O2RESOURCE`** and **`HOME`** environment variables; and the value of the *`context_name`* argument of the most recently executed **`lk_prologue`** function.

The files to read in are determined as follows:

If **`O2RESOURCE`** is defined, $O_2$Look attempts to load the file

*`O2_installation_directory`*`/`**`look/$O2RESOURCE/`***`context_name`***

If **`O2RESOURCE`** is not defined, $O_2$Look tries

*`O2_installation_directory`*`/`**`look/`***`context_name`***

$O_2$Look then looks for

**$HOME/.LKdefaults**

If neither 1 nor 2 exists, no resource file is loaded. If they both exist, they are both loaded, and any conflicting resource definitions are resolved by giving **.LKdefaults** priority.

Thus an application can change its look and/or behavior entirely, merely by issuing an **lk_prologue** function with a new application name; in the same way, an end-user may choose a whole new set of resources by changing his/her **O2RESOURCE** variable.

For example, one user might set the variable to **english** and another to **french**; each would run the same application, but see the prompts, tuple attribute names and so on in the language of their choice.

From the programmer's point of view, the only action required in order to create multilingual applications is the copying of the resource file from one **look** subdirectory to another, and the translation of the relevant character strings.

Resources are specified in these files in the same manner that other X Window or Motif resources are defined in files such as **.Xdefaults**: Each resource is specified in a single line of text, consisting of a resource name followed by a colon, then one or more spaces, and then a value for that resource:

*resource_name: resource_value*

If a resource specification is too long to fit on one line, it may be broken with the back-slash character (\); a back-slash at the end of a line signals that the next line is just a continuation of the present one. Similarly, the double-quote character (**"**) may appear in a resource specification only if it is preceded by a backlashes. In either case, the backlashes character does not appear in the specification itself; it is only a signal. An example:

**tour_booking.reservation.name.foregroundColor: red**

This resource specification identifies a very particular sub-mask of one single presentation: "In the **tour**_booking application, there is a presentation called **reservation** which displays a tuple value, one of whose attributes is called **name**. Display the value of that attribute in red."

The correspondence between the elements of the resource name and the presentations and masks on the screen is established by the *ed_name*argument of the $O_2$Look functions **lk_present**, **lk_refresh_presentation** and the mask functions given later in this section. The *ed_name* argument supplies a name for the widget (mask) to which the resource is to be applied. In order for $O_2$Look to associate

resources from resource files to masks, *ed_name* must be supplied — that is, not left null or blank. The meaning of each of the parts (called *editors*[1]) of the resource name given above is as follows:

- **tour_booking** is the name of the application, as defined in the **context_name** argument of the **lk_prologue** function.
- **reservation** is the name of a presentation, as given in the **ed_name** argument of the **lk_present** function which defined that presentation.
- **name** is the name of a sub-mask of the **reservation** presentation. Note that the **lk_present** function which defined the presentation could not have used an **lk_generic** or **lk_protected** mask, since those default masks do not allow the naming of specific tuple field sub-masks. The mask must have been **lk_tuple**, and one of the sub-masks of the **lk_tuple** mask was no doubt an **lk_atom** mask whose **ed_name** argument was **"name"**.
- **foregroundColor** is the name of an $O_2$Look resource, one of whose legal values is **red**. This is one of the resources available with the **lk_atom** mask.

Instead of identifying the names of individual applications, presentations and sub-masks, a resource specification may identify *classes*[2] of applications, presentations and sub-masks.

Under the X Window System, an editor class name is the same as the editor name except that its first letter is capitalized.

A resource specification may also use wild-card characters (*****) to indicate that *all* possible presentations or sub-masks are affected. For example:

```
TravelAgency*Tuple*Integer.backgroundColor: LightSteelBlue
```

This resource specification may be read, “In all applications that have to do with a travel agency, wherever there is a tuple with an integer component, display the integer component on a light steel blue background.”

The meaning of each part of this specification is:

- **TravelAgency** is the class name of the application, as defined in the *context_class* argument of the **lk_prologue** function. (Again, we are dealing with X Window classes here, and not with $O_2$ classes.)

1. The term editor can be misleading: An editor is simply any presentation, mask or sub-mask that displays data on the screen, regardless of whether the display allows editing (changing) of the data.
2. The term “class” here has nothing to do with O2 classes.

- **Tuple** is the class name of all **lk_tuple** masks. The asterisk in front of **Tuple** means that the specification applies to *all*[1] tuple displays in the **Travel_agency** group of applications, regardless of the presentation name.
- **Integer** is the class name of all integer masks (that is, **lk_atom** masks that display integers). The asterisk in front of **Integer** means that *all* integer components of tuple displays are affected, even if there are other sub-masks in between the tuple and the integer (for example, a tuple one of whose attributes is a set of integers).
- **backgroundColor** is the name of an $O_2$Look resource available for integer masks, and one of whose legal values is **Light**SteelBlue.

## Special resources for application icons

When an $O_2$ application is launched under $O_2$Look, the system defines and maps a presentation containing an icon representing the application. You can override this behavior by specifying a program named **dashboard** for the application. Refer to the $O_2$C Reference manual.

The menu associated with this icon holds the public programs of the application from which the end user may select. The appearance of both the application icon and its presentation may be tailored through the specification of graphic resources in a resource file.

This is done using the following information:

- At the invocation of the application, the system automatically performs an **lk_prologue** function, using as its *context_name* argument the name of the schema under which the application is operating, followed by an underscore character, followed by the name of the application:

  *schema_name_appli_name*

  Therefore, relevant resource specifications could be placed in an application-specific resource file called *context_name* under the **look** subdirectory of the $O_2$ installation directory.
- All of the presentation resources are available for the application icon presentation. The *ed_name* of this presentation is always "application"; so, for an application named **agency** operating under a schema called **sch1**, the following resource specification would determine a presentation title for the icon display:

1. Or, rather, it applies to all tuple masks which have an editor name. Any tuple masks whose ed_name argument is zero or null will remain unknown to the resource manager; neither the Tuple class specification nor the use of the wild card (*) will rescue such masks from their anonymity. The same applies to any nameless presentation, mask or sub-mask.

```
*sch1_agency.application.titleString:   Just testing...
```

- All of the resources for iconic representations of objects are available to the application icon itself; the *ed_name* of the icon is the name of the application. Continuing the above example, if you want to display the program menu in red on a yellow background, the following resource specifications would do so:

```
*sch1_agency.application.agency.menuForegroundColor: red
*sch1_agency.application.agency.menuBackgroundColor: yellow
```

## 4.3 Resource arrays

The graphic resources specified in resource files (or left at their default values) may be overridden at run-time by calls to various $O_2$Look functions.

This may be done either when a presentation or mask is created (by passing a list of resources to the function that creates it), or later (through the **lk_refresh_presentation** function).

In either case, the caller must supply two arguments: a count of the number of resources to be customized (generally called *rcount*) and a pointer to an array of resource specifications (usually called *resources*).

Each element in the array must be of type **Lk_resource**, which is known to the system as the following C structure:

```
typedef struct {
char* name;
char* value;
} Lk_resource;
```

The **name** string identifies the resource, and the **value** string indicates what it is to be set to.

***Note***

The **name** string is not the full name specification required for a resource file; it is only the last part of that specification, such as **"fontList"** or **"foregroundColor"**.

This is because the function itself already knows which presentation or mask is to be affected by the change.

The $O_2$Look functions that accept resource arrays are:

- ***lk_prologue***
  resource specifications affect all presentations under the newly-initiated application context. Only global resources may be specified here.

- ***lk_present***
  resource specifications affect only the presentation being created.

- ***lk_refresh_presentation***
  resource specifications affect only the presentation being refreshed. The resources are the same as for **`lk_present`**.

- ***All mask functions***
  resource specifications affect only the mask being created. However, this mask may be used as a sub-mask in several presentations at once.

  The resources that may be customized vary from one type of mask to another; the relevant list is given for each mask function described in the present section.

## 4.4 How to read resource tables

The following sections deal with tables of graphic resources which can be customized for certain masks or under certain conditions.

The tables have four columns:

- ***Name***
  The resource name.

- ***Type***
  The resource type.

- ***Default***
  The default value of the resource.

- ***Access***
  How the resource can be accessed.

This section now details how to read the graphic resource tables given in this chapter.

## Name

The resource name is given in full and is case sensitive.

In a resource array argument of an $O_2$Look function, this is the name (in full and alone) that must be assigned to the **`name`** string of an **`Lk_resource`** structure.

In a resource file, this name must be preceded by one or more editor names, specifying the presentation or mask whose resource is to be altered.

A resource name always begins with a lower-case letter; the name of the resource *`class`* is always the resource name with the first letter capitalized.

Immediately following each resource table, every resource name listed in the table is explained briefly.

## Type

The type of the resource is useful mainly to understand the kinds of values that the resource may take.

***Note***

In an **`Lk_resource`** structure, the **`value`** attribute is always a **`char *`**.

The following type specifications are used in resource values:

- ***Boolean***

  A Boolean value accepts the strings **"true"**, **"false"**, **"yes"**, **"no"**, **"on"** and **"off**", and is case insensitive.

- ***Font***

  A Font value is a string describing the properties of the font (rather the name of the file in which the font is stored).

  The string

  **"-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1"**

  specifies the foundry (adobe), the font family (courier), weight (bold), slant (oblique), set width (normal), size of the font in pixels (10), size of the font in tenths of a point (100), horizontal resolution (75 dpi), vertical resolution (75 dpi), spacing (m for mono-space), average width in tenths of pixel (60) and character set (iso8859-1).

  Since typing such names is not always desirable, wild card characters are acceptable.

  Any unnecessary part of a font name can be “wildcarded” by a question mark for any single character, and an asterisk for any group of characters. For example:

  **"*courier-bold-o-*-100-*"**

  As a general rule, you should always specify the following parts:

  - Font: Charter, Courier, New Century Schoolbook, Symbol, Times)
  - Weight: medium, bold
  - Angle: roman, italic, oblique
  - Size: 8, 10, 12, 14, 18, 24

  The rest may be wild-carded. The fonts are stored in the directories:

  **/usr/lib/X11/fonts/75dpi** and **/usr/lib/X11/fonts/100dpi**

- ***Color***

  A Color value may be either a string or a hexadecimal number. A set of predefined colors is assigned to specific (but not necessarily intuitive) names.

  This set is defined is the **rgb.txt** file located in **/usr/lib/X11**. You can also specify colors more exactly using a hexadecimal color string.

  The model used is the RGB model and the intensity of each primary color is controlled by a three-part digital value like **#RGB**, **#RRGGBB**, **#RRRGGGBBB** and **#RRRRGGGGBBBB** where **R**, **G** and **B** represent simple hexadecimal digit values.

For example: **#000** means black, **#FF0000** red, **#00F** blue, and **#FF00FF** magenta.

The default value is computed dynamically depending on the hardware:

- ***Pixmap***
  The Pixmap value is the full path name of the file containing a bitmap in X11 (or X10) format.

- ***C and $O_2$ types***
  The usual C and $O_2$ types are used as well: **int**, **short**, **string**, **char ***; **unsigned char** generally represents an enumerated type. Again, *all* resource values are represented as strings in $O_2$Look functions, regardless of their titular types.

## Default

The default value is taken by the resource if it is not otherwise specified.

## Access

This shows under what conditions the resource may be set.

- ***C***
  This resource can be set when the presentation is created and cannot be changed afterwards.
- ***S***
  The resource may be changed subsequent to creation. The **lk_refresh_presentation** function accomplishes this at the presentation level, for that presentation and all of its sub-masks.

  The list of global resources has no Access column, since global resources may be set only at the initialization of an application context, and have nothing to do directly with presentations or masks.

## 4.5 Global resources

Table 4.1 lists the graphic resources you can customize for an application context, either in a resource file associated to the *context_name* and/ or *context_class* argument of the **lk_prologue** function, or directly in the *resources* argument of **lk_prologue**.

Table 4.1 Global resources

| Name | Type | Default |
|---|---|---|
| presSpacing | short | 0 |
| highlightWidth | short | 3 |
| highlightColor | Color | Dynamic |
| cursorForegroundColor | Color | Dynamic |
| cursorBackgroundColor | Color | Dynamic |
| clipUse | Boolean | False |
| clipInitialState | int | 1 |
| clipIcon | Pixmap | UNSPECIFIED_PIXMAP |
| clipLabel | string | "LKClip" |
| clipGeometry | string | NULL |
| clipForegroundColor | Color | Dynamic |
| clipBackgroundColor | Color | Dynamic |
| clipNextType | unsignedchar | STRING |
| clipNextString | string | "Next" |
| clipNextPixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipInsensitiveNextPixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipPreviousType | unsignedchar | STRING |
| clipPreviousString | string | "Previous" |
| clipPreviousPixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipInsensitivePreviousPixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipDeleteType | unsignedchar | STRING |
| clipDeleteString | string | "Delete" |
| clipDeletePixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipInsensitiveDeletePixmap | Pixmap | UNSPECIFIED_PIXMAP |
| clipFontList | string | "fixed" |
| largeLowerBound | int | 50 |

- ***presSpacing*** Pixel spacing between presentations when using the placement argument with the x and y parameters of **lk_map** (e.g. **LK_RIGHT**, **LK_LEFT**). Horizontally, the system uses **presSpacing + x**, and vertically, **presSpacing + y**.
- ***highlightWidth*** Pixel width of highlighted border of mask during cut, copy or share (if a paste operation is allowed on that mask).
- ***highlightColor*** Color of highlighted border of mask during cut, copy or share operations (if a paste operation is allowed on that mask).
- ***cursorForegroundColor*** Foreground color of special cursor during cut, copy or share operations.
- ***cursorBackgroundColor*** Background color of special cursor during cut, copy or share.
- ***clipUse*** Is the Clipboard used or not
- ***clipInitialState*** Initial Clipboard state. Use 0 for a full window, or 1 for an icon
- ***clipIcon*** Clipboard icon bitmap.
- ***clipLabel*** Title of Clipboard window or icon
- ***clipGeometry*** Initial position and size of Clipboard window as specified in the string: **"=widthxheight+x_position+y_position"** , e.g. **"=200x200+300+300"**.
- ***clipForegroundColor*** Clipboard foreground color.
- ***clipBackgroundColor*** Clipboard background color.
- ***clipNextType*** Is the Clipboard Next button labeled with a string (**STRING**) or a bitmap (**PIXMAP**).
- ***clipNextString*** Next button string if **clipNextType** is **STRING**.
- ***clipNextPixmap*** Active Next button bitmap if **clipNextType** is **PIXMAP**.
- ***clipInsensitiveNextPixmap*** Inactive Next button bitmap, if **clipNextType** is **PIXMAP**.
- ***clipPreviousType*** Is Previous button labeled with a string or a bitmap.
- ***clipPreviousString*** Previous button string if **clipPreviousType** is **STRING**.
- ***clipPreviousPixmap*** Active Previous button bitmap if **clipPreviousType** is **PIXMAP**.
- ***clipInsensitivePreviousPixmap*** Inactive Previous button bitmap if **clipPreviousType** is **PIXMAP**.
- ***clipDeleteType*** Is the Clipboard Delete button labeled with a string (**STRING**) or a bitmap (**PIXMAP**).
- ***clipDeleteString*** Delete button string, if clipDeleteType is STRING.
- ***clipDeletePixmap*** Active Delete button bitmap, if **clipDeleteType** is **PIXMAP**.
- ***clipInsensitiveDeletePixmap*** Inactive Delete button bitmap, if **clipDeleteType** is **PIXMAP**.
- ***clipFontList*** Clipboard font.
- ***largeLowerBound*** Minimum set and list size above which display is by page .

## 4.6 Presentation resources

The following table lists the graphic resources which may be customized for an individual presentation, either in a resource file in association with the *ed_name* argument of either the **lk_present** or **lk_refresh_presentation** function, or directly in the *resources* argument of those functions.

Table 4.2 Presentation resources

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| titleFontList | Font | Dynamic | CS |
| buttonFontList | Font | Dynamic | CS |
| titleString | string | "" | CS |
| pen | unsigned char | DEFAULT | C |
| penType | unsigned char | PIXMAP | CS |
| penString | string | "save" | CS |
| penPixmap | Pixmap | Dynamic | CS |
| eraser | unsigned char | DEFAULT | C |
| eraserType | unsigned char | PIXMAP | CS |
| eraserString | string | "erase" | CS |
| eraserPixmap | Pixmap | Dynamic | CS |
| lockedPenType | unsigned char | PIXMAP | CS |
| lockedPenString | string | "locked" | CS |
| lockedPenPixmap | Pixmap | Dynamic | CS |
| lockedEraserType | unsigned char | PIXMAP | CS |
| lockedEraserString | string | "locked" | CS |
| lockedEraserPixmap | Pixmap | Dynamic | CS |

# 4 Graphic Resources

The graphic resources for presentations are :

- ***foregroundColor*** Presentation foreground color.
- ***backgroundColor*** Presentation background color.
- ***editable*** Can the presentation contents be modified by the end-user? If **`False`**, the user cannot change the values displayed. If **`True`**, he/she can. You can also set this resource for any component mask of the presentation, thereby overriding presentation's overall setting.
- ***replaceProcess*** Can presentation contents be replaced using paste? For an object presentation, note that replacing an object implies updating its encapsulated value and replacing its object identity.
- ***titleFontList*** Presentation title font.
- ***buttonFontList*** Font for the presentation buttons.
- ***titleString*** Presentation title compound string.
- ***pen*** Whether the pen button is displayed for the presentation. If this resource is **`INVISIBLE`**: the pen button is never displayed, if **`VISIBLE`**: it always visible, and if **`DEFAULT`**: it only appears if the presentation displays an object or is locked by **`lk_wait`** or **`lk_grab`**.
- ***penType*** Pen button type (**`STRING`** or **`PIXMAP`**).
- ***penString*** Pen button string if **`penType`** is **`STRING`**.
- ***penPixmap*** Pen button bitmap if **`penType`** is **`PIXMAP`**.
- ***eraser*** Is the eraser button displayed in the presentation. The options are **`VISIBLE`** or **`DEFAULT`** (eqivalent), and **`INVISIBLE`**.
- ***eraserType*** Eraser button type (**`STRING`** or **`PIXMAP`**).
- ***eraserString*** Eraser button string if **`eraserType`** is **`STRING`**.
- ***eraserPixmap*** Eraser button bitmap if **`eraserType`** is **`PIXMAP`**.
- ***lockedPenType*** Pen button type (**`STRING`** or **`PIXMAP`**) when the presentation is locked.
- ***lockedPenString*** Pen button string if the presentation is locked and **`lockedPenType`** is **`STRING`**.
- ***lockedPenPixmap*** Pen button bitmap if the presentation is locked and **`lockedPenType`** is **`PIXMAP`**.
- ***lockedEraserType*** Eraser button type (**`STRING`** or **`PIXMAP`**) when the presentation is locked.
- ***lockedEraserString*** Eraser button string is the presentation is locked and **`lockedEraserType`** is **`STRING`**.
- ***lockedEraserPixmap*** Pen button bitmap if the presentation is locked and **`lockedEraserType`** is **`PIXMAP`**.

## 4.7 Graphic resources for integers

An example window that uses the default mask for integers looks like this:

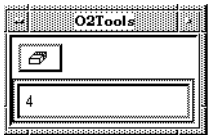



Table 4.3 lists the graphic resources which may be customized for an integer mask with the **lk_atom** function, either in a resource file in association with the *ed_name* argument, or directly in the *resources* argument.

Table 4.3 Graphic resources for integers

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | Font | "fixed" | CS |
| maxLength | int | MAXINT | CS |
| columns | short | 20 | CS |

The graphic resources of the mask for integers are as follows:

- ***foregroundColor*** Foreground color.
- ***backgroundColor*** Background color.
- ***editable*** Can end-user change the mask contents?
- ***replaceProcess*** Can mask contents be replaced using paste?
- ***shareProcess*** Can mask contents be copied?
- ***fontList*** Font used.
- ***maxLength*** Maximum length of integer string that can be entered from the keyboard.

- ***columns*** Initial width of integer mask in characters.

## 4.8 Graphic resources for real numbers

The following example uses the default mask for real numbers:

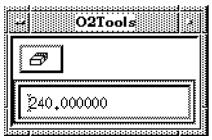



Table 4.4 lists the graphic resources which may be customized for an real number mask with the **lk_atom** function, either in a resource file in association with the *ed_name* argument, or directly in the *resources* argument.

Table 4.4 Graphic resources for real numbers

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | Font | "fixed" | CS |
| maxLength | int | MAXINT | CS |
| columns | short | 20 | CS |
| precision | short | 6 | CS |

The graphic resources of the mask for real numbers are as follows:

- ***foregroundColor*** Foreground color.
- ***backgroundColor*** Background color.
- ***editable*** Can end-user change the mask contents.
- ***replaceProcess*** Can the contents of the mask be replaced using paste.

- ***shareProcess*** Can mask contents be copied?
- ***fontList*** Font used.
- ***maxLength*** Maximum length of real number string entered via the keyboard.
- ***columns*** Initial width of the real number mask in characters.
- ***precision*** Number of digits after the readix separator.

## 4.9 Graphic resources for characters

The following window uses the default mask for characters:

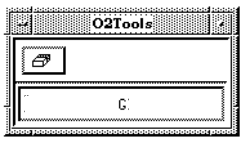



Table 4.5 lists the graphic resources which may be customized for a character mask with the **lk_atom** function, either in a resource file in association with the *ed_name* argument, or directly in the *resources* argument.

Table 4.5 Graphic resources for characters

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | Font | "fixed" | CS |

The graphic resources of the mask for characters are as follows:

- ***foregroundColor*** Foreground color.
- ***backgroundColor*** Background color.

- ***editable*** Can end-user change the mask contents?
- ***replaceProcess*** Can mask contents be replaced using paste?
- ***shareProcess*** Can mask contents be copied?
- ***fontList*** Font used.

## 4.10 Graphic resources for strings

The default mask for strings looks as in the following example window:

Table 4.6 lists the graphic resources which may be customized for a string mask with the **`lk_atom`** function, either in a resource file in association with the *`ed_name`* argument, or directly in the *`resources`* argument.

Table 4.6 Graphic resources for strings

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | Font | "fixed" | CS |
| maxLength | int | MAXINT | CS |
| columns | short | 20 | CS |

The graphic resources of the mask for strings are as follows:

- ***foregroundColor*** Foreground color.

- ***backgroundColor*** Background color.
- ***editable*** Can end-user change the mask contents?
- ***replaceProcess*** Can mask contents be replaced using paste?
- ***shareProcess*** Can mask contents be copied?
- ***fontList*** Font used .
- ***maxLength*** Maximum length of the string that can be entered from the keyboard.
- ***columns*** Initial width of the string mask measured in character spaces.

## 4.11 Graphic resources for boolean values

A boolean mask can consist of two parts: an “indicator” such as those pictured here (a diamond or square box that is filled if the boolean value is **`true`** and empty if **`false`**), and a label in the form of a string or bitmap.

The user may click on either of these parts to reverse the boolean value.

If a bitmap label is used, it too changes with the value; however, if a string label is used, it does not change.

These formats are controlled by graphic resources in the table below.

Table 4.7 lists the graphic resources which may be customized for a boolean mask with the **`lk_atom`** function, either in a resource file in association with the *`ed_name`* argument, or directly in the *`resources`* argument.

Table 4.7 Graphic resources for boolean values

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | string | "Fixed" | CS |

Table 4.7 Graphic resources for boolean values

| Name | Type | Default | Access |
|---|---|---|---|
| indicatorOn | Boolean | True | CS |
| indicatorType | unsigned char | DIAMOND | CS |
| selectColor | Color | Dynamic | CS |
| labelType | unsigned char | STRING | CS |
| labelString | string | "" | CS |
| selectPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| unselectPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |

The graphic resources of the mask for booleans are as follows:

- ***foregroundColor*** Foreground color.
- ***backgroundColor*** Background color.
- ***editable*** Can end-user change the mask contents?
- ***replaceProcess*** Can mask contents be replaced using paste?
- ***shareProcess*** Can mask contents be copied?
- ***fontList*** Font used .
- ***indicatorOn*** Can toggle indicator be drawn to the left of the label text or bitmap.
- ***indicatorType*** Shape of the toggle indicator. Permitted values are **DIAMOND** and **SQUARE**.
- ***selectColor*** Color that fills the center of the toggle indicator when it is set (**true**).
- ***labelType*** Label type. If **STRING**, the text of labelString is displayed; if **PIXMAP**, the bitmap specified by selectPixmap or unselectPixmap is displayed.
- ***labelString*** String to display when labelType is **STRING**.
- ***selectPixmap*** Bitmap to be used if labelType is **PIXMAP** and the boolean editor is selected (**true**).
- ***unselectPixmap*** Bitmap to be used if labelType is **PIXMAP** and the boolean editor is unselected (**false**).

## 4.12 Graphic resources for lists

The list default mask is a one-dimensional vertical array of element masks.

You use the graphic resources in association with the **lk_list** function either in a resource file in the *ed_name* argument or directly in the *resources* argument.

The following window uses the list mask:

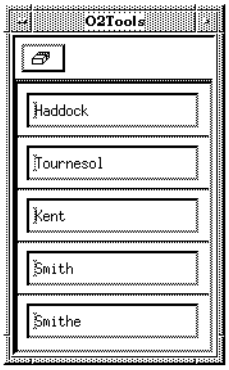



Table 4.8 lists the resources for lists.

The number of elments in a large list exceeds the limit set in the global resource **largeLowerBound**. These lists also have special resources. Table 4.9 the special resources for large lists.

Table 4.8 Graphic resources for lists

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| cutProcess | Boolean | True | CS |

Table 4.8 Graphic resources for lists

| Name | Type | Default | Access |
|---|---|---|---|
| insertProcess | Boolean | True | CS |
| newProcess | Boolean | True | CS |
| menuForegroundColor | Color | Dynamic | CS |
| menuBackgroundColor | Color | Dynamic | CS |
| menuTitleString | string | "New" | CS |
| menuFontList | Font | "fixed" | CS |
| orientation | unsigned char | VERTICAL | CS |
| grid | Boolean | True | CS |
| gridType | unsigned char | SHADOW_ETCHED_IN | CS |
| cellMaxHeight | short | 1000 | CS |
| cellMaxWidth | short | 1000 | CS |
| cellMinHeight | short | 10 | CS |
| cellMinWidth | short | 10 | CS |
| spacing | short | 10 | CS |
| visibleRowCount | short | 20 | C |

The specification of each resource is given below.

The graphic resources of the mask for lists are as follows:

- ***foregroundColor*** Foreground color.
- ***backgroundColor*** Background color.
- ***editable*** Can list value be edited. If False, you cannot add or remove elements from the list nor delete or replace the entire list.
- ***replaceProcess*** Can the contents of any of the list elements be replaced using paste. For an object replacing an object means updating its encapsulated value and replacing its object identity with that of the pasted object.
- ***shareProcess*** Can mask contents be shared; for list values, this is the same as copying.
- ***cutProcess*** Can you cut operation list elements using Cut.
- ***insertProcess*** Can you insert an element into the list using paste?
- ***newProcess*** Can you insert a new empty element into the list using the insert pop-up menu. (For an object list, this generates a new nil object and invokes new).

- ***menuForegroundColor*** Insert pop-up menu foreground color.
- ***menuBackgroundColor*** Insert pop-up menu background color.
- ***menuTitleString*** String displayed in the insert pop-up menu.
- ***menuFontList*** Font used in the insert pop-up menu.
- ***orientation*** List orientation. If **VERTICAL**, list elements one under another. If **HORIZONTAL**, the list elements are side by side.
- ***grid*** Are the lines separating the list elements (the grid) visible or not.
- ***gridType*** Style of the lines forming the grid. Possible values are:
  - **SINGLE_LINE, DOUBLE_LINE,** or **NO_LINE**
  - **SINGLE_DASHED_LINE,** or **DOUBLE_DASHED_LINE**
  - **SHADOW_ETCHED_IN** (double line etched into the window) or
  - **SHADOW_ETCHED_OUT** (double line etched coming out from the window)
- ***cellMaxHeight*** Maximum height of a single element cell.
- ***cellMaxWidth*** Maximum width of a single element cell.
- ***cellMinHeight*** Minimum height of a cell. (minimum height £ maximum height).
- ***cellMinWidth*** Minimum width of a cell. (minimum width £ maximum width).
- ***spacing*** Space in pixels between two elements in two adjacents cells.
- ***visibleRowCount*** Number of visible rows at creation.

If the number of elments in your list exceeds the limit set in the global resource **largeLowerBound**, then the list is a large list. These large lists also have special resources given in Table 4.9.

Table 4.9 Special resources for large lists

| Name | Type | Default | Access |
|---|---|---|---|
| buttonBarForegroundColor | Color | Dynamic | CS |
| buttonBarBackgroundColor | Color | Dynamic | CS |
| buttonForegroundColor | Color | Dynamic | CS |
| buttonBackgroundColor | Color | Dynamic | CS |
| nextType | unsigned char | STRING | CS |
| nextString | string | "Next" | CS |
| nextPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| insensitiveNextPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| previousType | unsigned char | STRING | CS |

Table 4.9 Special resources for large lists

| Name | Type | Default | Access |
|---|---|---|---|
| previousString | string | "Previous" | CS |
| previousPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| insensitivePreviousPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| fontList | Font | "fixed" | CS |

The specification of each resource and the graphic resources of the mask for large lists are as follows:

- ***buttonBarForegroundColor*** Button bar foreground color.
- ***buttonBarBackgroundColor*** Button bar background color.
- ***buttonForegroundColor*** Foreground color of the button bar buttons.
- ***buttonBackgroundColor*** Background color of the buttons in the button bar.
- ***nextType*** Is button used to display next page using **STRING** or **PIXMAP**.
- ***nextString*** Button label for next page display when **nextType** is **STRING**.
- ***nextPixmap*** Button bitmap for next page display when **nextType** is **PIXMAP** and button is active.
- ***insensitiveNextPixmap*** Button bitmap for next page display, when button is inactive and **nextType** is **PIXMAP**.
- ***previousType*** Is button used to display previous page using **STRING** or **PIXMAP**.
- ***previousString*** Button label for previous page display when **previousType** is **STRING**.
- ***previousPixmap*** Button bitmap for previous page display when that button is active and **previousType** is **PIXMAP**.
- ***insensitivePreviousPixmap*** Button bitmap for previous page display when button is inactive and **previousType** is **PIXMAP**.
- ***fontList*** Font used for the button labels.

## 4.13 Graphic Resources for sets

The set default mask is a two-dimensional array of element masks.

The graphic resources are used with lk_set, either in a resource file with the *ed_name* argument, or directly in the *resources* argument.

The following window includes the default set mask:

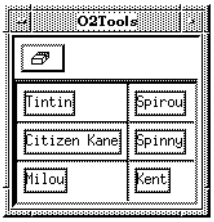



Table 4.10 lists the graphic resources for sets.

The elements of a large set exceed the limit specified in the global resource **largeLowerBound**).

In addition to the resources listed above for set masks, large set have the following resources. Table 4.11 list the special resources for large sets.

Table 4.10 Graphic Resources for sets

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| cutProcess | Boolean | True | CS |
| insertProcess | Boolean | True | CS |
| newProcess | Boolean | True | CS |
| menuForegroundColor | Color | Dynamic | CS |

Table 4.10 Graphic Resources for sets

| Name | Type | Default | Access |
|---|---|---|---|
| menuBackgroundColor | Color | Dynamic | CS |
| menuTitleString | string | "New" | CS |
| menuFontList | Font | "fixed" | CS |
| orientation | unsigned char | VERTICAL | CS |
| grid | Boolean | True | CS |
| gridType | unsigned char | SHADOW_ETCHED_IN | CS |
| cellMaxHeight | short | 1000 | CS |
| cellMaxWidth | short | 1000 | CS |
| cellMinHeight | short | 10 | CS |
| cellMinWidth | short | 10 | CS |
| spacing | short | 10 | CS |
| visibleRowCount | short | 20 | C |
| initNbColumns | int | 2 | C |
| visibleColumnCount | short | 20 | C |

The graphic resources of the mask for sets are as follows:

- ***foregroundColor*** Foreground color for the set.
- ***backgroundColor*** Background color for the set.
- ***editable*** Can the set value be edited. If False, the elements cannot be added to or removed from the set; the set canot be deleted or replaced in its entirety.
- ***replaceProcess*** Can you replace set contents using paste. For a set of objects, replacing an object implies updating its encapsulated value and replacing its identity with that of the pasted object.
- ***shareProcess*** Can mask contents be shared; for set values, this is the same as copying.
- ***cutProcess*** Can a cut operation be performed on elements of the set.
- ***insertProcess*** Can you insert an element into the set with a paste operation.
- ***newProcess*** Can you insert a new empty element into the set with new process. (For an object set this generates a new nil object and invokes the new process on it.)
- ***menuForegroundColor*** Insert pop-up menu foreground color.

- ***menuBackgroundColor*** Insert pop-up menu background color.
- ***menuTitleString*** String displayed in the insert pop-up menu.
- ***menuFontList*** Font used in the insert pop-up menu.
- ***orientation*** Set orientation. If **VERTICAL**, each set column is displayed vertically, if **HORIZONTAL**, horizontally.
- ***grid*** Is grid visible (grid lines separate set elements).
- ***gridType*** Style of the lines forming the grid. Possible values are:
  - **SINGLE_LINE, DOUBLE_LINE,** or **NO_LINE**
  - **SINGLE_DASHED_LINE,** or **DOUBLE_DASHED_LINE**
  - **SHADOW_ETCHED_IN** (double line etched into the window) or
  - **SHADOW_ETCHED_OUT** (double line etched coming out from the window)
- ***cellMaxHeight*** Maximum height of a single element cell.
- ***cellMaxWidth*** Maximum width of a single element cell.
- ***cellMinHeight*** Minimum height of a cell. Less than or equal to the maximum height.
- ***cellMinWidth*** Minimum width of a cell. Less than or equal to the maximum width.
- ***spacing*** Pixel space between two elements placed in two adjacents cells. The blank frame around the elements is half this number of pixels thick.
- ***visibleRowCount*** Number of visible rows at creation. If this number is greater than or equal to the number of elements in the set divided by the number of visible columns
- ***initNbColumns*** Total number of columns in the set mask at creation time.
- ***visibleColumnCount*** Number of visible columns at creation time. If this number is greater than or equal to the actual number of columns (**initNbColumns**, all columns are visible; otherwise, a scroll bar appears.)

If the elements of a set exceed the limit specified in the global resource **largeLowerBound**), your set is a large set. :

Table 4.11 Special resources for large sets

| Name | Type | Default | Access |
|---|---|---|---|
| buttonBarForegroundColor | Color | Dynamic | CS |
| buttonBarBackgroundColor | Color | Dynamic | CS |
| buttonForegroundColor | Color | Dynamic | CS |
| buttonBackgroundColor | Color | Dynamic | CS |
| nextType | unsigned char | STRING | CS |

Table 4.11 Special resources for large sets

| Name | Type | Default | Access |
|---|---|---|---|
| nextString | string | "Next" | CS |
| nextPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| insensitiveNextPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| previousType | unsigned char | STRING | CS |
| previousString | string | "Previous" | CS |
| previousPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| insensitivePreviousPixmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| fontList | Font | "fixed" | CS |

The graphic resources of the mask for large sets are as follows:

- ***buttonBarForegroundColor*** Foreground color of the button bar.
- ***buttonBarBackgroundColor*** Background color of the button bar.
- ***buttonForegroundColor*** Foreground color of the buttons in the button bar.
- ***buttonBackgroundColor*** Background color of the buttons in the button bar.
- ***nextType*** Is button for next page display identified by **STRING** or **PIXMAP**.
- ***nextString*** Button label for next page display, when nextType is **STRING**.
- ***nextPixmap*** Button bitmap for next page display when button is active and **nextType** is **PIXMAP**.
- ***insensitiveNextPixmap*** Button bitmap for next page display when button is inactive and **nextType** is **PIXMAP**.
- ***previousType*** Is button used for previous page display identified by **STRING** or **PIXMAP.**
- ***previousString*** Button label for previous page display when **previousType** is **STRING**.
- ***previousPixmap*** Button bitmap for previous page display when button is active and **previousType** is **PIXMAP**.
- ***insensitivePreviousPixmap*** Button bitmap for previous page display when button is inactive and **previousType** is **PIXMAP.**
- ***fontList*** Font used for the button labels.

## 4.14 Graphic resources for tuples

The following window gives an example of the tuple mask:

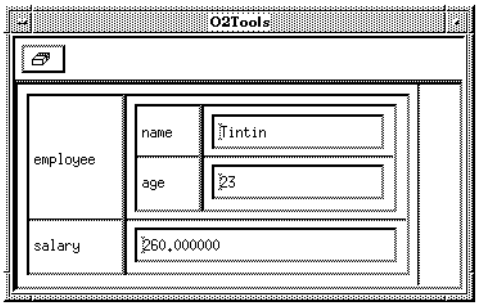



Table 4.12 lists the graphic resources which may be customized for a tuple mask, either in a resource file in association with the *ed_name* argument, or directly in the *resources* argument of the function.

Table 4.12 Graphic resources for tuples

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| fontList | Font | Dynamic | CS |
| orientation | unsigned char | VERTICAL | CS |
| labelList | char * | NULL | CS |
| labelAlignment | unsigned char | ALIGNMENT_BEGINNING | CS |
| labelVerticalPosition | unsigned char | CENTER | CS |
| labelBoxWidth | short | 0 | CS |
| labelBoxHeight | short | 0 | CS |
| attributeMaxWidth | short | 1000 | CS |
| attributeMaxHeight | short | 1000 | CS |
| visibleRowCount | short | 10 | C |
| grid | Boolean | True | CS |
| notEditableAttributeList | char * | NULL | CS |

- ***foregroundColor*** Tuple foreground color.
- ***backgroundColor*** Tuple background color.
- ***editable*** Can the tuple value be changed? If False, you cannot delete the tuple or replace it in its entirety, and you cannot change any of its attributes.
- ***replaceProcess*** Can the contents of any of the tuple attributes be replaced using paste. For an object attribute, replacing an object implies updating its encapsulated value and replacing its object identity with that of the pasted object.
- ***shareProcess*** Can mask contents be shared; this is the same as copying.
- ***fontList*** Font for the tuple labels.
- ***orientation*** Orientation of the tuple attribute display (**VERTICAL** or **HORIZONTAL**).
- ***labelList*** Text of the attribute labels. This is a newline-separated list:

  **label1\nlabel2\n...labelN**

  The labels used, by default, are the schema attribute names.
- ***labelAlignment*** Label alignment type. The following are the possible alignment types:
  - **ALIGNMENT_BEGINNING:** left-justified labels (the default)
  - **ALIGNMENT_CENTER:** centered labels
  - **ALIGNMENT_END:** right-justified labels
- ***labelVerticalPosition*** Specifies the vertical position of the labels in the label boxes. The following are vertical position types:
  - **TOP:** labels at the top of the box
  - **CENTER:** labels centered in the box (default)
  - **BOTTOM:** labels on the bottom line of the box
- ***labelBoxWidth*** Label box width (in pixels) when orientation is **VERTICAL**. If zero, the size of the tuple's widest box is used for all of the attributes.
- ***labelBoxHeight*** Label box height (in pixels) when orientation is **HORIZONTAL**. If zero, the size of the tuple's tallest box is used for all of the attributes.
- ***attributeMaxWidth*** Maximum width of tuple attributes.
- ***attributeMaxHeight*** Maximum height of tuple attributes.
- ***visibleRowCount*** Specifies the number of visible attributes at creation time. If this number is greater than or equal to the number of displayable attributes, all attributes will be visible; otherwise, scroll bars will appear.
- ***grid*** Are the lines that separate attributes (grid) displayed.
- ***notEditableAttributeList*** Which tuple attributes have read-only values. The is a newline-separated list:

  **attribute1\nattribute2\nattributeN**

  Each attribute named is protected against any user changes. Note that the attribute names used are defined in database schema for the tuple, and not necessarily the attribute labels that appear on the screen. The screen labels can be changed using the **labelList** resource (above).

## 4.15 Graphic Resources for Objects

The list of graphic resources which may be customized for an object, either in a resource file in association with the *ed_name* argument, or directly in the *resources* argument of the function, is the same as that available with the **lk_object** mask.

You can represent an object in four different ways:

- ***expanded*** A fully-expanded presentation shows the full or part of the object value.
- ***iconic*** An iconic presentation of an object is a box with the object class name centered in the box. You can add a bitmap to the icon either by redefining the bitmap method for the class concerned or by specifying the bitmap resource for the object mask.
- ***nil object*** If there is no object to be displayed as yet, a nil object presentation is used. A nil object is a nonexistent object — i.e., one without an object identity. The class of a nil object is known, but the object has not yet been created (in $O_2$C, with a new instruction or by an assignment; or under $O_2$Look, with the new process). For example, when you create a new tuple value, one of whose attributes is an object. The default representation of a nil object is a black box.
- ***undefined object*** An undefined object is a nonexistent object of an unspecified or incompletely-specified class. An undefined object differs from a nil object in that it cannot be created, or until its class is properly defined. The default representation is a question mark inside a box.

The following table lists the graphic resources you use to customize an object mask.

You specify these resources using either the **lk_object** or **lk_protected** function, and give them either in a resource file using the *ed_name* argument, or directly in the *resources* argument of the function.

Table 4.13 Graphic resources for objects

| Name | Type | Default | Access |
|---|---|---|---|
| foregroundColor | Color | Dynamic | CS |
| backgroundColor | Color | Dynamic | CS |
| editable | Boolean | True | CS |
| replaceProcess | Boolean | True | CS |
| shareProcess | Boolean | True | CS |
| titleFontList | Font | "Fixed" | CS |

Table 4.13 Graphic resources for objects

| Name | Type | Default | Access |
|---|---|---|---|
| titleString | string | NULL | CS |
| menuBackgroundColor | Color | Dynamic | CS |
| menuForegroundColor | Color | Dynamic | CS |
| menuFontList | Font | Dynamic | CS |
| menuTitleString | string | "methods" | CS |
| menu | char * | NULL | C |
| methodResult | Boolean | False | CS |
| bitmap | Pixmap | UNSPECIFIED_PIXMAP | CS |
| newString | string | "new" | CS |
| newProcess | Boolean | True | CS |

- ***foregroundColor*** Object mask foreground color.
- ***backgroundColor*** Object mask background color.
- ***editable*** Can user change the object representation or its encapsulated value.
- ***replaceProcess*** Can you replace object mask contents using paste. This refers to the encapsulated value of the object, not to the object itself. This resource therefore only applies to expanded object representations.

  However, the **`replaceProcess`** resource of a presentation or mask that contains the object (e.g. a set of objects) determines whether the object can be replaced using paste. Replacing an object means updating its encapsulated value and replacing its object identity.
- ***shareProcess*** Can the object be shared?
- ***titleFontList*** Object mask title font for iconic and expanded representations of a real object, but not representations of nil or undefined objects.
- ***titleString*** Object mask title string. The default is the string returned by the $O_2$Look method title, which unless redefined, is the object class name. This resource applies to iconic or expanded real object representations but not to nil or undefined object representations.
- ***menuBackgroundColor*** Methods menu background color for iconic or expanded real object representations, or the new pop-up menu for a nil object. It has no effect on representations of undefined objects.
- ***menuForegroundColor*** Methods menu foreground color for iconic or expanded representations of a real object, or the new pop-up menu for a nil object. It has no effect on representations of undefined objects.
- ***menuFontList*** Methods menu font for iconic or expanded representations of a real object, or the new process menu for a nil object. It has no effect for representations on undefined objects.

- ***menuTitleString*** Pull-down menu title when the default menu is used (ie. menu resource is **NULL**).

- ***menu*** Contents of the public methods menu fo iconic or expanded representations of a real object, but not representations of nil or undefined objects. The menu specification is hierarchical and has the syntax:

```
{ {"entry1" "item1" method_name1 ... "itemN" method_nameN}
  {"entry2" "item1" method_name1 ... "itemP" method_nameP}
    ...
  {"entryM"  ... }
 }
```

## Important

The menu resource can only be set when you create the presentation. You cannot change it afterwards.

If you use the word **separator** before any item specification, you generate a horizontal separator line in the displayed menu.

In an expanded object mask, the menu hierarchy is a menu bar with each entry name shown in the bar. You then select an entry using the left mouse button which then pulls down a menu displaying the menu items. In an icon object mask, there is only one pull-down menu which, if it has only one entry, shows the various item names of that entry. If the menu has several entres, it shows the various entry names, each followed by an arrow. If select an entry displays a sub-menu of entry items.

If the **menu** resource specifies an erroneous menu and at run-time the user tries to select a non-existent method, $O_2$Look gives a warning.

This resource does not effect nil or undefined objectsrepresentations. If specified, this resource overrides the function of the $O_2$Look method **menu**. If not specified, a single-entry menu is provided by the **menu** method which, by default, lists the public methods of the object class.

If you use mnemonics, the menu specification is hierarchical with the syntax:

```
{
{"entry1" [mnemonic-name]
     "item1" [item1-mnemonic] [item1-accelerator] method-
name1
     "item2" [item2-mnemonic] [item2-accelerator] method-
name2
       ...
}
  {"entry2" [mnemonic-name]
       ...
}
} For example:
{
{ "Compile"  _C
          "foo"      _f Alt<key>F8 Foo
       "bar"      _b            Bar
       "new ..."                New
   }
  { "Display"
     "display"  _d            Display
   }
}
```

A mnemonic is only one letter preceded by a underscore. It gives you an alternate means of activating a button when it is visible. When a mnemonic is specified, you must use the Alt key while pressing the mnemonic. You can activate the button by pressing either the shifted or the unshifted mnemonic key. An accelerator can activate a visible or invisible button. It is a string that describes a set of modifier (Alt, Ctrl) and the key that may be used to select the button. The key is introduced by the keyword **key**.

- ***methodResult*** Is the returned value of each method invoked interactively on this object should displayed on the screen in an $O_2$Look presentation. This can be useful for debugging.
- ***bitmap*** Object mask bitmap. This resource does not apply to expanded objects. The default, if this resource is not specified, is a black box for nil objects; a question mark inside a box for undefined objects; and for iconic representations of objects, the bitmap value returned by the $O_2$Look method bitmap.

- ***newString***

  String of the pop-up menu used to invoke the **new** process on a nil object. This resource applies only to nil object representations.

- ***newProcess***

  Is the **new** process, which invokes a constructor on the nil object, allowed. If true, a pop-up menu appears when the user clicks the right mouse button on the nil object mask. If the user then selects the single entry on this menu (whose text is defined with **newString** above), then the $O_2$ constructor for the class of the nil object is invoked to create a new object, and a presentation appears in which the user is invited to define the new object's data content. This resource applies only to nil object representations.

# 4 Graphic Resources

# 5 Widget Look

This chapter describes the $O_2$ Widget Look.

This is an X Toolkit compatible user-interface component. It contains all the power of $O_2$Look in a simple widget.

It has the following sections:

- Introduction
- Tools

# 5 Widget Look

## 5.1 Introduction

To face the technical challenge of designing and implementing a graphical user interface, $O_2$ has three tools,

- $O_2$Look, an integrated powerful tool enabling the graphical display and manipulation of any object in the $O_2$ database.
- The $O_2$ - Xt, specifically designed for user interfaces developed with an X Toolkit. It is a programming library tailored to the special requirements of user interface construction within an $O_2$ database application. For details refer to the documentation "Using X toolkit with $O_2$".
- $O_2$Look widget, used in conjunction with $O_2$Look or the $O_2$ - Xt library. This is an X Toolkit compatible user interface component. The $O_2$Look widget contains all the power of $O_2$Look in a simple widget.

The rest of this chapter now describes the $O_2$Look widget classes.

## 5.2 Tools

This section describes the following classes

- **`Xo2Look`**
- **`Xo2CreateLook`**
- **`Xo2SetO2Value`**
- **`Xo2ConsultO2Value`**

## Xo2Look

**Summary** The **Look** widget class

**Syntax** **#include "Look.h"**

**Arguments** None

**Description** **Look** is a manager you use to encapsulate an $O_2$Look presentation inside a widget in order to integrate an $O_2$Look presentation in a widget hierarchy.

It uses geometry management so that the widget size matches the $O_2$Look presentation size when it is displayed directly by $O_2$Look.

Notice that after being initialized, the size of the Look widget cannot be changed by a request from its child. Its father is the only one that can resize it.

**Classes** **Look** inherits behavior and resources from the **Core**, **Composite**, **Constraint** and **XmManager** classes.

The class pointer is **xo2LookWidgetClass**.

The class name is **Xo2Look**.

**Translations** **Xo2Look** inherits translations from XmManager.

**New resources** Table 5.1 defines the set of widget resources you use to specify data.

You can also set the resource values for the inherited classes to set the attributes for this widget.

The access column codes in the following tables indicate whether resource can be set at creation time (C) or using **XtSetValues**(S), whether its can be retrieved using **XtGetValues** (G), or is applicable or not (N/ A).

Table 5.1 **Xo2Look resource set**

| Name | Type | Default | Access |
|---|---|---|---|
| Xo2Nvalue | void * | dynamic | CS |
| Xo2Nkind | Xo2ValueKind | dynamic | CS |
| Xo2Nmask | int | dynamic | CS |
| Xo2NeditorName | String | dynamic | CS |
| Xo2NpresResourcesCount | int | 0 | CS |
| Xo2NpresResources | void * | NULL | CS |

These resources are specified below.

- ***Xo2Nvalue*** — Specifies the $O_2$ value or object to be displayed inside the Look Widget. The value given in this resource must be first cast to (**o2 void**).

## Note

When using the Look widget in $O_2C$, the $O_2$ value given to **XtSetArg** cannot be an expression; it must be an $O_2C$ variable.

The following expression is incorrect:

```
XtSetArg(mt_args[mt_n_args], Xo2Nvalue,\
(o2 void)(*MyList)[0]);
```

The correct expression is:

```
i = (*MyList)[0];
XtSetArg(mt_args[mt_n_args], Xo2Nvalue, (o2 void)i);
```

## Important

Only $O_2$ values or objects are allowed as **o2** values. Constant strings or (**char***) variables are forbidden.

- ***Xo2Nkind*** — This is the kind of value is specified in the **Xo2Nvalue** resource. **Xo2Nvalue** can have the following values:
  - **XO2_INTEGER** - an integer value.
  - **XO2_CHAR** - a char value.
  - **XO2_REAL** - a real value.
  - **XO2_STRING** - a string value.
  - **XO2_BYTES** - a bytes value.
  - **XO2_BOOLEAN** - a boolean value.
  - **XO2_VALUE** - a complex $O_2$ value.
  - **XO2_OBJECT** - an $O_2$ object.

- ***Xo2Nmask*** Specifies the $O_2$Look mask used to display the $O_2$ value given in **`Xo2Nvalue`**.
- ***Xo2NeditorName*** A string representing the $O_2$Look presentation that corresponds to the $O_2$ value given in **`Xo2Nvalue`** and can be used in a resource file to specify graphical resources for this presentation.
- ***Xo2NpresResourcesCount*** An integer count of the number of $O_2$Look resources given in **`Xo2NpresResources`**.
- ***Xo2NpresResources*** A pointer to an array of graphic resources to be customized in the mask given in **`Xo2Nmask`** (unless **`Xo2NpresResourcesCount`** is zero).

**Inherited resources**

**Look** inherits behavior and resources from the following superclasses.

For a complete description of each resource, refer to the OSF/ Motif programmer's reference manual.

The inherited resources include three resource sets: **`XmManager`**, **`Composite`** and **`Core`**.

Table 5.2 gives the resource set for **`XmManager`**.

Table 5.2 **XmManager resource set**

| Name | Type | Default | Access |
|---|---|---|---|
| XmNbottomShadowColor | Pixel | dynamic | CSG |
| XmNbottomShadowPixmap | Pixmap | XmUNSPECIFIED_PIXMAP | CSG |
| XmNForeground | Pixel | dynamic | CSG |
| XmNhelpCallback | XtCallbackList | NULL | C |
| XmNhighlightColor | Pixel | dynamic | CSG |
| XmNhighlightPixmap | Pixmap | dynamic | CSG |
| XmNnavigationType | XmNavigationType | XmTAB_GROUP | CSG |
| XmNshadowThickness | Dimension | dynamic | CSG |
| XmNstringDirection | XmStringDirection | dynamic | CG |
| XmNtopShadowColor | Pixel | dynamic | CSG |
| XmNtopShadowPixmap | Pixmap | dynamic | CSG |
| XmNtraversalOn | Boolean | True | CSG |
| XmNunitType | unsigned char | dynamic | CSG |
| XmNuserData | Pointer | NULL | CSG |

Table 5.3 gives the **`Composite`** resource set.

Table 5.3 **Composite resource set**

| Name | Type | Default | Access |
|---|---|---|---|
| XmNchildren | WidgetList | NULL | G |
| XmNinsertPosition | (*)() | NULL | CSG |
| XmNnumChildren | Cardinal | 0 | G |

Table 5.4 gives the **Core** resource set.

Table 5.4 **Core resource set**

| Name | Type | Default | Access |
|---|---|---|---|
| XmNaccelerators | XtAccelerators | NULL | CSG |
| XmNancestorSensitive | Boolean | dynamic | G |
| XmNbackground | Pixel | dynamic | CSG |
| XmNbackgroundPixmap | Pixmap | XmUNSPECIFIED_PIXMAP | CSG |
| XmNborderColor | Pixel | XtDefaultForegroud | CSG |
| XmNborderPixmap | Pixmap | XmUNSPECIFIED_PIXMAP | CSG |
| XmNborderWidth | Dimension | 0 | CSG |
| XmNcolormap | Colormap | dynamic | CG |
| XmNdepth | int | dynamic | CG |
| XmNdestroyCallback | XtCallbackList | NULL | C |
| XmNheight | Dimension | dynamic | CSG |
| XmNinitialResourcesPersistent | Boolean | True | C |
| XmNmappedWhenManaged | Boolean | True | CSG |
| XmNscreen | Screen * | Dynamic | CG |
| XmNsensitive | Boolean | True | CSG |
| XmNtranslations | XtTranslations | NULL | CSG |
| XmNwidth | Dimension | dynamic | CSG |
| XmNx | Position | 0 | CSG |
| XmNy | Position | 0 | CSG |

## Xo2CreateLook

**Summary** Creates the Look widget.

**Syntax**

```
#include "Look.h"
Widget Xo2CreateLook (parent, name, arglist, argcount)
Widget parent;
String name;
ArgList arglist;
Cardinal argcount;
```

**Arguments**

| | |
|---|---|
| *parent* | Specifies the parent widget ID |
| *name* | Specifies the name of the created widget |
| *arglist* | Specifies the argument list |
| *argcount* | Specifies the number of attribute/ value pair in the argument list |

**Description** **Xo2CreateLook** creates an instance of a Look widget and returns the associated widget ID.

Before a widget Look is created, the $O_2$Look **lk_proglogue** function must be called.

**Example**

```
method body create_look_widget(father_i: integer):integer
{
   Widget lw, father;
   Arg       mt_args[20];
   int       mt_n_args = 0;


   father = (Widget) father_i;
   lw = Xo2CreateLook(father, "_look", mt_args,
                      mt_n_args);
   return((o2 integer) lw);
}
```

## Xo2SetO2Value

**Summary** Assigns a $O_2$Look value to a Look widget.

**Syntax**

```
#include "Look.h"
void Xo2SetO2Value (lw, kind, o2value, mask, ed_name,
rcount, resources)
Xo2LookWidget lw;
Xo2ValueKind kind;
void *o2value;
int mask;
char *ed_name;
int rcount;
Lk_resource *resources;
```

**Arguments**

| | |
|---|---|
| *lw* | An $O_2$Look widget |
| *kind* | Specifies the type of $O_2$ value |
| *o2value* | Specifies the $O_2$ value. |
| *mask* | The mask identifier. |
| *ed_name* | String representing mask name. |
| *rcount* | Integer count of the number of special resources given in the *resources* argument. |
| *resources* | A pointer to an array of graphic resources to cbe customized in the mask. |

**Description** This function creates a presentation of the $O_2$ value *o2value* using the *mask*, *ed_name*, *rcount* and *resources* arguments.

It inserts the top widget of this presentation as the child of the $O_2$Look widget *lw*.

The *o2value* must be cast in **(o2 void)**.

***Important***

Only $O_2$ values or objects are allowed as **o2** values. Constant strings or (**char***) variables are forbidden.

**Example** An example is given below.

```
method body text_to_widget(father_i:integer,
text:Text):integer

{

   Widget lw, father;
   Arg       mt_args[20];
   int       mt_n_args = 0;

   father = (Widget) father_i;
   lw = Xo2CreateLook(father, "_look", mt_args,
                  mt_n_args);

   Xo2SetO2Value(lw, XO2_OBJECT, (o2 void) text,
                 lk_object("otext", 0, 0
                           lk_specific("text",
                                       0,0,"text",0,0)),
                 "text_test", 0, NULL);
   XtManageChild(lw);
   return(lw);

}
```

## Xo2ConsultO2Value

**Summary** Updates a database object according to a Look widget content.

**Syntax**

```
#include "Look.h"
void Xo2ConsultO2Value (lw, kind, address)
Xo2LookWidget lw;
Xo2ValueKind kind;
void *address;
```

**Arguments**

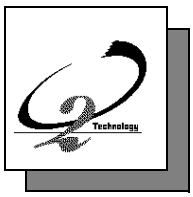
| | |
|---|---|
| *lw* | An $O_2$Look widget |
| *kind* | Specifies the type of $O_2$ value |
| *address* | Address $O_2$C variable. |

**Description** The first argument of the function is the $O_2$Look widget you want to consult, the second argument is the kind of the $O_2$ value currently contained in the widget and the third argument is the address of an $O_2$C variable cast in **(o2 void)** when using the function in $O_2$C and is the address of a **Handle** when using this function elsewhere.

**Example**

```
method body consult_text_widget(lwi:integer): Text
{
   o2 Text t = new Text;
   Widget lw = (Widget) lwi;
   Xo2ConsultO2Value(lw, XO2_OBJECT, &(o2 void) t);
   return(t);
}


method body consult_int_widget(lwi:integer): integer
{
   o2 integer i;
   Widget lw = (Widget) lwi;
   Xo2ConsultO2Value(lw, XO2_integer, &i);
   return(i);
}
```




# INDEX

## A

**all_presentations** 27
alpha
    command 67

## B

**bitmap** 41
Boolean graphic resources **119**

## C

C 13
C types 24
C++
    Interface 13
Callback methods 40, 48
Character graphic resources **117**
**Composite** 139
**Constraint** 139
**consult** 28
Copying elements 18
**Core** 139

## D

Data
    Types **74**
**disable_method** 29
**display** method 31
Display methods 25–??
**display** option 66
Drag and drop **18**

## E

**edit** 33
**enable_method** 35
Environment variable
    **DISPLAY** 66
    **HOME** 67
    **O2RESOURCE** 67
**erase** 43
**erase_all** 36

## F


Functions
- Introduction 20
- Library 49–72
- **lk_all_mapped_presentations** 50
- **lk_all_presentations** 51
- **lk_consult** 52
- **lk_current_presentation** 54
- **lk_delete_all** 56
- **lk_delete_presentation** 57
- **lk_epilogue** 58
- **lk_free** 59
- **lk_grab** 60
- **lk_map** 61
- **lk_present** 64
- **lk_prologue** 66
- **lk_refresh_presentation** 68
- **lk_unmap** 70
- **lk_wait** 71
- Masks **79–99**


## G


**GenPresentation** 64

Global resources **111–112**

Graphic resources **102–135**
- Boolean values **119**
- Characters **117**
- Large lists **123**
- Large sets **127**
- Lists **121–124**
- Objects **131–135**
- Real numbers **116**
- Sets **125–128**
- Strings **118**
- Tuples **129–130**


## H


**HOME** 67


## J


Java 13


## L


Library functions 49–72

List
- Large list 123

List graphic resources **121**

**lk_all_mapped_presentations** 50

**lk_all_presentations** 51

**lk_atom** 80

**Lk_attribute_mask** 24

**lk_consult** 52

**LK_CONTENT** option 62

**LK_COORDINATE** option 61

**lk_current_presentation** 54

**lk_delete_all** 56

**lk_delete_presentation** 57

**lk_epilogue** 58

LK_ERASE
- lk_free 59
- lk_grab 60
- lk_wait 71

LK_FAILED
- lk_grab 60
- lk_wait 71

**lk_free** 59
**LK_FREE** option 62
**lk_generic** 83
**LK_GEOMETRIC** option 62
**lk_grab** 60
**lk_list** 85
**LK_MANUAL** option 61
**lk_map** 61
**Lk_mask** 24
**lk_method** 87
**LK_MOUSE** option 61
**lk_object** 89
**lk_present** 64
**Lk_presentation** 24
**lk_prologue** 66
**lk_protected** 91, 92
**lk_refresh_presentation** 68
**Lk_resource** 24
**LK_RIGHT** option 61
LK_SAVE
  lk_free 59
  lk_grab 60
  lk_wait 71
**LK_SCREEN** argument 62
**lk_set** 94
**LK_SLAVE** option 62
**lk_specific** 96
**LK_STACK** option 61
**lk_tuple** 97
**lk_unmap** 70
**lk_wait** 71
**LKdefaults** 67

## *M*

Mask
  Functions **79–99**
  Introduction 20
  **lk_atom** 80
  **lk_destroy_mask** 82
  **lk_generic** 83
  **lk_list** 85
  **lk_method** 87
  **lk_object** 89
  **lk_protected** 92
  **lk_set** 94
  **lk_specific** 96
  **lk_tuple** 98
**menu** 44
Methods
  **all_presentations** 27
  **bitmap** 41
  Callback 40, 48
  **disable_method** 29
  Display 25–??
  **display** 31
  **edit** 33
  **enable_method** 35
  **erase** 43
  **erase_all** 36
  Introduction 19
  **menu** 44
  **present** 37
  **refresh_all** 38
  **rubout** 39
  **save** 46
  **title** 47
Moving elements 18

## N

nolook
Option 25

## O

$O_2$
Product line 12
**o2_look.h** 25
$O_2$C 13
$O_2$C
Options 25
$O_2$Corba 13
$O_2$DBAccess 13
$O_2$Engine 12
$O_2$Graph 13
$O_2$Kit 13
$O_2$Look 13
$O_2$ODBC 13
**O2RESOURCE** 67
$O_2$Store 12
$O_2$Tools 13
$O_2$Web 13
Object
Graphic resources **131–135**
Option
**display** 66
**lk_map** 61
**nolook** 25
$O_2$C 25
OQL 13

## P

**present** 37
Presentation
Definition **17**
Resources **113–114**
Programming 19

## R

Real number resources 116
**refresh_all** 38
Resource
Arrays **106–107**
Files **102–106**
Tables **107–110**
Resource specification 76
Resources
Boolean values **119**
Characters **117**
Global **111–112**
Introduction 20, **102**
Large lists **123**
Large sets **127**
Lists **121–124**
Objects **131–135**
Presentation **113–114**
Real numbers **116**
Sets **125–128**
Strings **118**
Tuples **129–130**
**rubout** 39

## S

**save** 46
Set
  Graphic resources **125–128**
  Large sets **127**
String graphic resources **118**

## T

**title** 47
Tuple
  Graphic resources **129–130**
Type
  C 24
  Data **74**

## V

Variable
  **DISPLAY** 66
  **HOME** 67, **102**
  **O2RESOURCE** 67, **102**

## X

**XmManager** 139
**XO2_BOOLEAN** 140
**XO2_BYTES** 140
**XO2_CHAR** 140
**XO2_INTEGER** 140
**XO2_OBJECT** 140
**XO2_REAL** 140
**XO2_STRING** 140
**XO2_VALUE** 140
**Xo2ConsultO2Value** 146
**Xo2CreateLook** 143
**Xo2Look** 139
**xo2LookWidgetClass** 139
**Xo2NeditorName** 141
**Xo2Nkind** 140
**Xo2Nmask** 141
**Xo2NpresResources** 141
**Xo2NpresResourcesCount** 141
**Xo2Nvalue** 140
**Xo2SetO2Value** 144
**XtGetValues** 139
**XtSetArg** 140
**XtSetValues** 139